UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

Simulación Numérica del Flujo de Materiales Granulares

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO MECANICO ELECTRICISTA

(AREA MECANICA)

P R E S E N T A

GUSTAVO JOSEPH GONZALEZ

DIRECTOR DE TESIS: DR. BALTASAR MENA INIESTA

México, D. F. Junio, 1997

A Boru,
Chucho,
Oci,
Pule y
Udi.

Agradecimientos

A la Universidad Nacional Autónoma de México, por permitirme, con su gran mosaico de actividades, escuelas, foros y gente, obtener una formación tanto académica como cultural.

A la Facultad de Ingeniería y a su Programa de Alto Rendimiento Académico, por los años de apoyo y las oportunidades ofrecidas para conocer gente, aprender cosas nuevas, aplicar cosas viejas y disfrutar de las cosas cotidianas.

A Fundación UNAM, por proporcionar dieciocho meses de beca para el servicio social y la elaboración de esta tesis.

Al Instituto de Investigaciones en Materiales, por la infraestructura, equipo, libros, conferencias, donas, café y galletas, así como también por el apoyo económico prestado durante mi estancia en él.

A Baltasar, por haber propuesto tanto el tema como mi asistencia al Congreso Internacional de Reología; también, por proporcionar ideas, conciertos, música y comidas que tanto complementaron la labor granular.

A Enrique Geffroy, quien más que ser el profesor "de al lado", se convirtió en una verdadera fuente de ideas y guía para lograr el mejor funcionamiento de esta simulación numérica.

Al Dr. Otis Walton, por su profunda, desinteresada y completa explicación de `3dshear`.

A todo el equipo de trabajo del Laboratorio de Reología, pero en especial a los Brolos (Omar y Edgar), quienes lograron hacer de ésta una labor tanto amena como productiva, en la que a diario se aprendía algo.

A Adán (y Sol), Boli, Cespín (y asociados), Gaby, Juanita, Julito, Lore, Robert T. (y anexos), Sol (y Adán), Toñito y demás banda: cinco años maravillosos, muchos más por venir.

Contenido

Tabla de Ilustraciones

Objetivos e Introducción

Hipótesis: **El algoritmo de dinámica molecular propuesto por el Dr. Otis Walton es capaz de simular exitosamente el flujo dentro del silo hexagonal inventado por el Dr. Baltasar Mena.**

El presente trabajo pretende ser una primera aproximación a la simulación completa de las propiedades de flujo dentro de un silo hexagonal, desarrollado por el Dr. Baltasar Mena en el Instituto de Investigaciones en Materiales de la UNAM. El diseño actual del silo ha probado ser altamente ventajoso para la conservación adecuada de los granos alimenticios. El objetivo concreto de esta tesis es sentar las bases para poder analizar numéricamente el flujo granular dentro del silo. Para esto se realizaron modificaciones especiales a un algoritmo de dinámica molecular desarrollado por el Dr. Otis Walton en los Lawrence Livermore National Laboratories de los Estados Unidos. El objetivo a largo plazo de este estudio consiste en aprovechar las simulaciones numéricas para mejorar el diseño del silo hexagonal.

El estudio de los materiales granulares y su flujo ha cobrado gran importancia en los últimos años. Se han desarrollado diversas teorías para explicar su comportamiento, pero resulta difícil analizarlos en detalle porque prácticamente todos los métodos de medición requieren invadir de alguna u otra manera al medio granular, perturbándolo en mayor o menor medida. Una forma de analizar el interior de un flujo granular consiste en hacer una simulación numérica satisfactoria. Si se logra aproximar la geometría de uno de estos flujos por medio de una simulación, podemos tener una relativa certeza de que las propiedades calculadas corresponderán a las propiedades reales (empaquetamiento, energía cinética, tensor de esfuerzos, etc.).

En el primer capítulo se describe el estado actual del conocimiento respecto a los materiales granulares, enfatizando que su comportamiento los vuelve, por derecho propio, un estado más

de la materia. En el segundo capítulo se hace un resumen de los métodos más comunes de simulación numérica de flujos, describiendo sobre todo la dinámica molecular, pues el sistema desarrollado se basa justamente en esta técnica.

De ese punto en adelante, se describe detalladamente el algoritmo propuesto por Walton, así como también las modificaciones realizadas. Como apoyo a cada una de dichas modificaciones, se presenta una serie de resultados de simulaciones que coinciden cada vez mejor con la realidad, para culminar con una simulación del siio completo.

En los apéndices se presentan los listados del código fuente del algoritmo de simulación, los archivos de entrada para cada corrida representativa, el listado del programa de postprocesamiento de datos y las referencias bibliográficas.

Este trabajo se llevó a cabo en su totalidad en el Laboratorio de Reología del Instituto de Investigaciones en Materiales de la UNAM, bajo la dirección del Dr. Baltasar Mena Iniesta.

Generalidades sobre Materiales Granulares

Breve historia de los granulares

¿Quién pudiera calcular la trayectoria de una molécula? ¿Cómo sabemos si la creación de los mundos no está determinada por la caída de granos de arena?[a] Fue con esta frase que Victor Hugo transpuso los límites del sentido común al relacionar la creación de los mundos con el movimiento de simples granos de arena. Y sin embargo, su metáfora demostró su real alcance y actualidad a poco más de cien años de haber sido planteada. Es difícil pensar en alguna industria que no emplee materiales granulares en sus procesos de transformación. La erosión, la dinámica de placas tectónicas y las avalanchas son todas procesos geológicos de carácter granular. Incluso el altero de libros, papel y demás objetos que casi todos tenemos en nuestros escritorios suelen estar tan cerca de su ángulo de reposo que cualquier perturbación aleatoria puede provocar una avalancha hacia el piso. En este capítulo trataré de mostrar la riqueza y complejidad que tienen en sí mismos estos importantes materiales.

Los materiales granulares son simples. Son grandes conglomerados de partículas macroscópicas individuales. En ausencia de efectos cohesivos, las fuerzas que actúan entre ellos son estrictamente repulsivas por lo que la forma del conjunto está determinada por sus fronteras exteriores y la acción de la gravedad. Cualquier gas intersticial que se encuentre presente puede ser fácilmente despreciado al evaluar las propiedades del material granular sin incurrir en prácticamente ningún error. Y a pesar de esta mustia apariencia, los materiales granulares se comportan de manera diferente a cualquier otra forma de agregación de la materia—sólidos, líquidos o gases—y pueden por derecho propio ser considerados un estado más de la materia.

[a] Victor Hugo (1802–1855), escritor francés, en *Les Misérables* (1862).

El meollo de este comportamiento único radica en dos características peculiares: la temperatura termodinámica no tiene efecto alguno y las interacciones entre los diversos granos son disipativas por causa de la fricción estática y la inelasticidad de las colisiones[1]. Un montón de arena que tenga una pendiente menor a su ángulo de reposo, por ejemplo, se comporta como un sólido: permanece en reposo aún cuando la acción gravitacional crea esfuerzos microscópicos en su superficie. Si inclinamos el montículo varios grados por arriba del ángulo de reposo de la arena, las partículas comenzarán a fluir. Sin embargo, este flujo difiere claramente del de un fluido ordinario pues sólo se presenta en la superficie del montículo (*Figura 1*), sin que exista movimiento alguno en el grueso de los granos. Podríamos entonces modelar el flujo granular como el de un gas denso, pues los gases también se conforman de partículas individuales cuya interacción cohesiva es despreciable. Pero a diferencia de los gases ordinarios, la energía térmica, kT, es absolutamente insignificante en los materiales granulares. El factor energético relevante para una partícula de masa m y diámetro d es su energía potencial mgd, donde g es la aceleración gravitacional. Para un grano de arena típico a temperatura ambiente, la energía potencial es por lo menos 10^{12} veces kT, lo que significa que cualquier relación termodinámica se vuelve inútil. El hecho de que $kT \approx 0$ implica que la entropía del sistema es varios órdenes de magnitud menor que los efectos dinámicos, por lo que algunos efectos que aparentemente violan el principio del incremento de la entropía se presentan con singular recurrencia en los sistemas granulares.

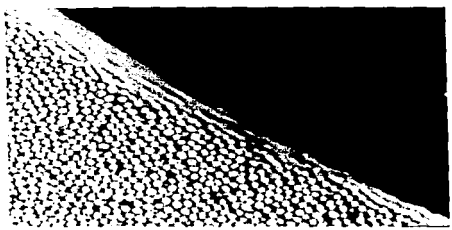


Figura 1. En esta toma se puede observar cómo el flujo de granos de mostaza se presenta solamente en la superficie, sin que el material por debajo del ángulo de reposo se vea afectado.

Por otra parte, al ser la temperatura termodinámica despreciable, los sistemas granulares están privados de experimentar el equivalente a un cambio de fase. Cada configuración metaestable de un material granular permanecerá indefinidamente a menos que una acción externa lo perturbe. Además, no existe la posibilidad de que se alcance un equilibrio termodinámico entre las configuraciones vecinas. Debido a que cada configuración es única, es difícil alcanzar la reproducibilidad del comportamiento granular, aún a grandes escalas y cerca del límite estático, donde la fricción se torna importante. Otro papel de la temperatura en los gases ordinarios es el

de proporcionar una velocidad característica a nivel microscópico. Los materiales granulares, de nueva cuenta, erradican completamente este papel y la única velocidad característica que se puede considerar es la impuesta por un flujo. Es posible modelar una "temperatura granular" en función de las fluctuaciones de velocidad en torno a la velocidad promedio del flujo.[2] Pero estas aproximaciones no siempre restablecen la termodinámica o la hidrodinámica del sistema debido a que las colisiones granulares son inelásticas.

La ciencia de los materiales granulares tiene toda una historia, con una vasta literatura dedicada a su entendimiento. En ella se encuentran consignados los nombres de científicos notables como Coulomb, quien postuló las ideas de la fricción estática o Faraday, quien descubrió el efecto convectivo que se presenta en los polvos sujetos a vibraciones. Reynolds, por ejemplo, introdujo el concepto de dilatancia, el cual señala que un material granular asentado debe sufrir una expansión (o dilatación) antes de fluir como respuesta a un esfuerzo cortante.[3] Pero ha sido en los últimos diez años que el estudio de estos materiales ha adquirido un creciente impulso dentro de la física. La idea de criticalidad cooperativa[a] es un concepto fundamental para la descripción de muchos sistemas dinámicos disipativos y tiene sus orígenes en las avalanchas que se generan en un montón de arena acomodado a un ángulo cercano al de reposo. Diversas analogías se presentan entre el flujo de granulares y el flujo de muchos fluidos convencionales. Así también, se observan fenómenos que están relacionados con situaciones tan distantes como la ruptura dieléctrica de los semiconductores o la dinámica sísmica.

La ciencia de los materiales granulares también se relaciona claramente con diversas industrias. Entre éstas se incluyen la farmacéutica, que se basa en el manejo de polvos y píldoras (tabletas, cápsulas y demás), la agricultura y la industria alimentaria, donde diario se manipulan y transportan granos, semillas y otros objetos semejantes o la industria de la construcción, que depende de la grava, la arena y el cemento. Otros procesos, como la fundición de piezas automotrices, dependen de la cuidadosa creación de moldes de arena compacta. Como se señala en diversos estudios,[4] hasta un 40% de la capacidad total de muchas industrias se desperdicia por problemas relativos al transporte de este tipo de materiales. Así pues, cualquier pequeño avance en el entendimiento actual de los materiales granulares podría redituar ahorros substanciales. Trataré

[a] En inglés, *self-organized criticality*.

ahora de hacer un bosquejo de las tres facetas más comunes que presentan los materiales granulares.

Un sólido poco común: arena en reposo

Los materiales granulares presentan, aún en reposo, toda una gama de comportamientos poco comunes. Por ejemplo, al ser contenidos en recipientes cilíndricos de altura considerable, como los silos verticales, la presión no es función de la profundidad, como en un fluido normal; es decir, que la presión en la base del contenedor no crece indefinidamente conforme la altura del material contenido aumenta. Al contrario, para una columna lo suficientemente alta, la presión alcanza un valor máximo que es independiente de la altura. Debido a las fuerzas de contacto entre los granos y a la fricción estática, las paredes del silo cargan el peso sobrante.

La distribución de fuerzas dentro de un montículo se puede visualizar fácilmente colocando una hoja de papel carbón en el fondo del recipiente y midiendo las áreas de las huellas dejadas en un papel por efecto de la fuerza, f, de cada una de las partículas individuales. La distribución de fuerzas es

$$P(f) = c\exp(-f / f_0)$$

donde c y f_0 son constantes.[5] Las fluctuaciones de f son de gran magnitud y son proporcionales a la profundidad, al igual que la fuerza promedio, en vez de ser proporcionales a la raíz cuadrada de la profundidad, como se hubiera esperado en un fluido convencional. Este comportamiento ha sido explicado en términos de un modelo simple en el que las partículas colocadas en una red distribuyen su peso de manera no uniforme y aleatoria sobre las partículas del nivel inmediatamente inferior. Las diversas soluciones exactas del modelo descrito, que concuerdan tanto con los experimentos como con las simulaciones, arrojan una distribución exponencial de las fuerzas.

Otro tema fundamental en la física de los materiales granulares tiene que ver con su empaquetamiento. Dependiendo de la manera en la que se llene un contenedor, un conjunto aleatorio de esferas puede quedar distribuido con fracciones de empaquetamiento en el rango comprendido entre $\eta = 0.55$ y $\eta = 0.64$.[6] Gracias a la fricción estática se forman cadenas de fuerza (*Figura 2*) que pueden mantener al conjunto en una configuración metaestable dentro de los límites mencionados. Pero, ¿cómo pasa el sistema de una de estas configuraciones a otra? Al ser kT despreciable, la única forma en la que se puede conseguir energía para lograr el cambio en densi-

dad es por medio de perturbaciones externas, *e.g.* vibraciones. Modelando como "temperatura" la compacticidad del material, se puede reemplazar a la energía por una función del volumen. De esta manera, la entropía del sistema sigue siendo el logaritmo del número de estados posibles para un volumen dado. Al introducir energía al sistema por medio de vibraciones, se logra destrabarlo de manera que puede cambiar lentamente de una configuración a otra.

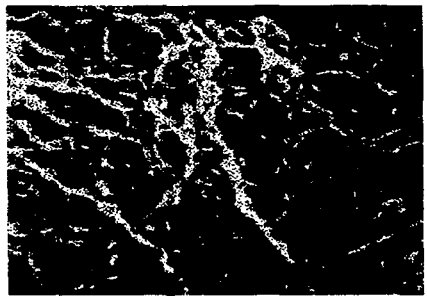


Figura 2. Para visualizar las cadenas de fuerza, se colocan esferas de vidrio un recipiente igualmente de vidrio. Los espacios intersticiales se rellenan con un fluido del mismo índice de refracción al del vidrio y se coloca el conjunto entre polarizadores cruzados (de tal forma que no pase luz). Al aplicar una carga uniforme a la superficie, se hacen evidentes las cadenas de esfuerzo como cambios en la polarización de la luz.

Estudios recientes de materiales granulares sometidos a vibraciones para lograr su asentamiento muestran que estos sistemas se relajan de una manera logarítmicamente lenta. [7] Se han observado sistemas que aún después de 100 000 ciclos de vibración siguen compactándose de manera importante. Aún cuando se han propuesto diversos modelos para tratar de explicar este comportamiento, una analogía sencilla para visualizarlo[8] consiste en pensar que el sistema es un estacionamiento público. Supongamos por un momento que en el estacionamiento no hay cajones trazados y que un gran número de automóviles iguales (léase partículas) se encuentra estacionado. Para una persona que desee estacionar un vehículo adicional (o insertar una partícula en el ensamble) normalmente sucede que, aunque amplios, los espacios disponibles entre los coches no son lo suficientemente grandes. La pregunta es: ¿cuántos de los demás automóviles (o partículas) deberán moverse ligeramente para que el vehículo adicional pueda caber? Si todo este proceso de compactado se lleva a cabo estacionando y "desestacionando" vehículos aleatoriamente, se requiere del movimiento cooperativo de muchos objetos (cantidad exponencialmente creciente con la densidad) para lograr abrir un nuevo cajón. Como resultado, la velocidad con la que se alcanza la densidad del estado estacionario es logarítmica en el tiempo.

Un líquido poco común: la hidrodinámica granular

De manera cualitativa, observamos que los materiales granulares pueden fluir como líquidos. Existen diversos modelos teóricos que describen este comportamiento. Se puede hablar entonces de una hidrodinámica granular (aún cuando no haya nada que moje) en el sentido de que estos modelos teóricos describen medios continuos empleando ecuaciones diferenciales de forma análoga a las ecuaciones de Navier-Stokes de los fluidos newtonianos. Sin embargo, los modelos que describen los flujos granulares no tienen la generalidad de las ecuaciones de Navier-Stokes, pues estas últimas se obtienen de promediar a lo largo de tiempos y distancias muchos órdenes de magnitud mayores que los que rigen a escala microscópica y muchos órdenes de magnitud menores a los que rigen en el flujo macroscópico. Desgraciadamente, esta gran separación de órdenes de magnitud es imposible de alcanzar en los flujos granulares.

Dos idealizaciones útiles para representar los flujos granulares son el flujo potencial lento y el flujo rápido de gases. Como los sistemas granulares disipan energía rápidamente, es común que presenten ambos tipos de flujo en regiones cercanas. Aún persiste la pregunta de cómo modelar la transición entre estos dos flujos. Se puede emplear la teoría cinética[9] para modelar el flujo de granulares a bajas densidades. Para mantener este estado se requiere, sin embargo, de un suministro constante de energía (vibratoria, por ejemplo).

El otro extremo se puede modelar por medio de deformaciones plásticas casiestáticas, basándonos en el principio de Reynolds de la dilatancia[10] y en la experiencia, que nos dicta que la deformación de polvos compactados es típicamente irreversible. La dilatancia se presenta debido a que las partículas de los materiales granulares típicamente se enclavan entre sí bajo la acción de un esfuerzo normal. La única manera de lograr un flujo ante la presencia de un esfuerzo cortante es que algunas de estas partículas enclavadas logren brincar por encima de otras partículas, lo que forzosamente conlleva a una dilatación del sistema.

Hablar de modelos específicos implica hablar de las leyes de conservación en su forma diferencial y de ecuaciones constitutivas. Así pues, se cuenta con una ecuación de continuidad para la conservación de la masa, una ecuación de la energía y una ecuación del momento. De estas tres, es la última la más interesante, pues relaciona el tensor de esfuerzos T_{ij} y el tensor de rapidez de deformación

$$\dot{\gamma}_{ij} = \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i},$$

donde v_i es la i-ésima componente de la velocidad. Uno de los modelos más sencillos[11] es

$$T_{ij} = \sigma\left(\delta_{ij} + k\frac{V_{ij}}{|V|}\right),$$

donde $|V|^2 \equiv \sum V_{ij}^2$ y k es una constante característica de cada material. Más específicamente, $k/\sqrt{2} = \text{sen}\,\delta$, donde δ es el ángulo de fricción interna. Si comparamos esta ecuación con la de Navier-Stokes observamos que el término viscoso, normalmente dependiente de la viscosidad y el gradiente de velocidad, es independiente de la rapidez de deformación. Esta característica es notable, pues implica que un incremento global en la velocidad no afecta la condición de esfuerzo.

El tipo de modelos descrito se emplea en la mecánica de suelos y en el diseño de equipo de proceso, como tolvas. Sin embargo, se ha descubierto que el flujo en tolvas angostas no concuerda con los modelos matemáticos. Por medio de imágenes de rayos X, se puede observar el flujo a través de una tolva para los casos de materiales de granos rugosos y de granos lisos. En el caso de los primeros, se presentan ondas viajeras de densidad, en contraste con la ausencia de éstas en los materiales tersos y casi esféricos. Las ondas se pueden propagar hacia arriba o hacia abajo, dependiendo de cuán pronunciado sea el ángulo de la tolva. Hace falta profundizar más en el estudio de la geometría del grano, la cual juega en la estructura del flujo un papel crítico y poco entendido.

Un aspecto apasionante de la física de los materiales granulares es la extraña gama de respuestas que presentan al estar sometidos a vibraciones. Actualmente existe un fuerte debate por tratar de explicar las causas de estas respuestas. Dos de las causas más buscadas son: (i) la de la convección y del amontonamiento inducidos por vibraciones al material y (ii) la de la segregación de tamaños inducida por vibraciones.

La primera persona en observar patrones convectivos en los materiales granulares sujetos a vibración fue Michael Faraday hace unos 160 años,[12] pero todo este tiempo no ha servido para entender cabalmente el mecanismo involucrado. Tanto la segregación como la convección ocurren cuando se agita al material en la dirección vertical. Típicamente, se hace oscilar al recipiente siguiendo una función sinusoidal $z = A\cos(\omega t)$. Cuando la aceleración máxima del contenedor alcanza valores mayores que la aceleración gravitacional—cuando $\Gamma \equiv A\omega^2/g$ es mayor que 1—

el material se separa del piso del recipiente durante una fracción de cada ciclo, lo cual le permite dilatarse y formar una recirculación convectiva macroscópica que continuamente acarrea granos. En un experimento típico que emplee recipientes cilíndricos o rectangulares se genera un flujo hacia arriba en el centro y hacia abajo en una estrecha franja cercana a las orillas, lo cual hace que se forme un montículo central desde el que de presenta una continua avalancha de partículas.[13] Es posible lograr un flujo hacia abajo en el centro e invertir el sentido de la convección inclinando las paredes del recipiente hacia afuera. En general, el efecto combinado de la forma del contenedor, la fricción entre las paredes y las partículas y las fronteras internas de fases se pueden combinar para modificar la dirección de la convección.[14]

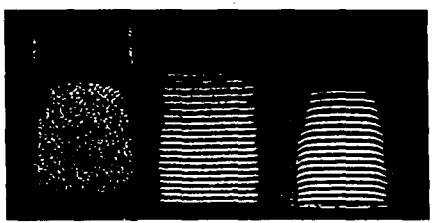


Figura 3. A la izquierda se puede observar la imagen de resonancia magnética, sin ningún patrón evidente, de granos de ajonjolí en reposo. Intencionalmente se adhirió algunos granos a las paredes del recipiente para darles visibilidad. La imagen del centro muestra los perfiles de densidad que se presentan con oscilaciones periódicas, mientras que para la imagen de la derecha se dio un solo impulso vertical "instantáneo".

Se han propuesto varios mecanismos para explicar estas rotaciones inesperadas en un experimento en el que se sacude uniformemente a todo el conjunto. Uno de ellos establece que el arrastre de partículas debido a la fricción con las paredes crea una estrecha capa que se desplaza rápidamente, forzando la circulación. En varios experimentos recientes se han determinado tanto el perfil de velocidades en la convección como su dependencia funcional con la profundidad por medio de imágenes de resonancia magnética para visualizar el flujo dentro del material sin la necesidad de introducir una sonda que perturbe al material (*Figura 3*).[15] En contra de lo que ocurre con los líquidos, aquí el flujo más rápido ocurre en una estrecha región cerca de las paredes, en vez de presentarse una condición de no deslizamiento. Este resultado complica fuertemente la forma en la que se deben considerar las condiciones de contorno para los materiales granulares.

Un segundo mecanismo, sugerido originalmente por Faraday, relaciona al fluido intersticial atrapado con la convección y la formación de montículos. La experimentación en este sentido está aún en desarrollo, con resultados contradictorios. De hecho, aún es un reto teórico la formulación de un modelo que involucre tanto la fricción como el efecto del gas intersticial.

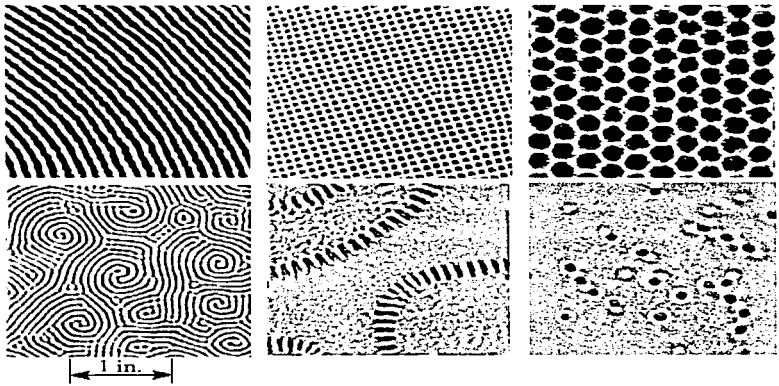


Figura 4. Se presentan vistas superiores de un lecho de 7-8 partículas de profundidad, al vacío y sujeto a vibraciones verticales. Para una frecuencia dada, existe una amplitud crítica a la cual la superficie cambia de plana a estriada. A 25 Hz, se crean patrones cuadrangulares, mientras que si la amplitud aumenta, los patrones se vuelven hexagonales y luego toman la forma de costuras en una pelota de béisbol. En la última imagen se muestra, para una profundidad de 17 partículas, un tipo de estructura conocido como *oscilón*.[16]

Por si fuera poco, la superficie libre de un material granular sujeto a vibraciones puede presentar tanto patrones irregulares y caóticos como fenómenos ondulatorios. Estas ondas pueden ser tanto estacionarias (cuando casi no se presentan montículos) como viajeras (para un material cuyos montículos crezcan rápidamente). En general, las ondas estacionarias que se forman son de carácter subarmónico. Los patrones de superposición de estas ondas tienen una semejanza impresionante con las inestabilidades de Faraday que se presentan en los fluidos comunes (*Figura 4*).

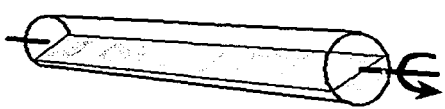


Figura 5. Después de un cierto número de rotaciones completas se observa una marcada segregación entre granos de arena de Ottawa (de color obscuro) y granos de sal (de color claro). Esta segregación se presenta para toda la altura y no sólo para la superficie.

En cuanto a la segregación de diferentes materiales, ésta se puede observar claramente en tambores rotatorios de longitud considerable y eje horizontal. En ellos se tiene que las partículas con diferentes ángulos dinámicos de reposo se congregan en regiones fuertemente delimitadas a lo largo del eje de rotación (*Figura 5*).[17] A veces se requiere lograr el efecto contrario y determinar cuán bien se mezclan partículas de diferentes tipos en un tambor rotatorio. En este caso la sucesión continuada de avalanchas superficiales produce el mezclado (*Figura 6*). El grado mezclado es función directa del nivel hasta el que se haya llenado el cilindro[18] y se puede calcular de manera puramente geométrica. Tanto el mezclado como la segregación tienen una importancia real en procesos como la separación de finos,[a] deseables o indeseables, o el mezclado de medi-

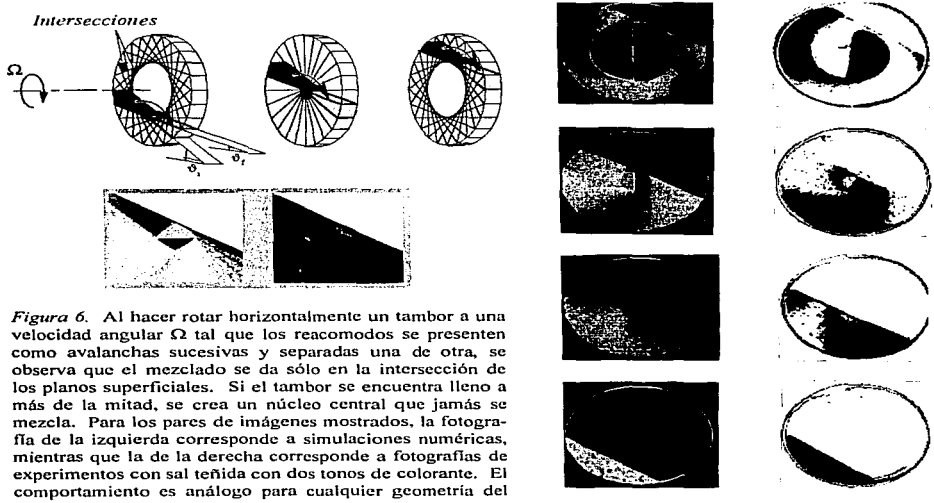


Figura 6. Al hacer rotar horizontalmente un tambor a una velocidad angular Ω tal que los reacomodos se presenten como avalanchas sucesivas y separadas una de otra, se observa que el mezclado se da sólo en la intersección de los planos superficiales. Si el tambor se encuentra lleno a más de la mitad, se crea un núcleo central que jamás se mezcla. Para los pares de imágenes mostrados, la fotografía de la izquierda corresponde a simulaciones numéricas, mientras que la de la derecha corresponde a fotografías de experimentos con sal teñida con dos tonos de colorante. El comportamiento es análogo para cualquier geometría del tambor.

[a] Minerales fracturados, en el caso de minería; tamo, en el caso de granos pulverizados.

camentos en polvo con un aglutinante, proceso en el que es fundamental alcanzar una mezcla homogénea y bien controlada.

Se ha tratado por medio de diversos mecanismos de explicar tanto la segregación como el mezclado de materiales granulares sujetos a vibraciones (*Figura 7*). Uno de estos mecanismos es el de cernido (partículas pequeñas cayendo a través de los intersticios de las partículas grandes) seguido de reacomodos locales (que hacen que la dilatación del conjunto cree espacios en los que las partículas pequeñas se precipitan, empujando hacia arriba a las partículas grandes). En el caso de vibraciones verticales, se ha hallado una relación directa de la segregación con la convección: las partículas grandes son arrastradas por el flujo hacia arriba, siendo obligadas a permanecer ahí debido a que literalmente no caben en la estrecha capa de flujo hacia abajo que se presenta cerca de las paredes.

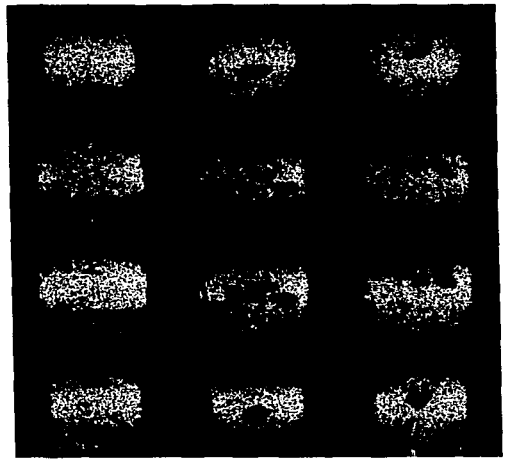


Figura 7. Se muestra por medio de resonancia magnética la segregación de granos de café (manchas obscuras) en una matriz de ajonjolí sujeta a oscilaciones verticales. El tiempo aumenta de izquierda a derecha. Las tres tomas superiores corresponden a un corte cercano a la parte frontal del recipiente, mientras que las inferiores corresponden a la parte posterior. Los dos juegos centrales son cortes cercanos a la mitad. Se observa cómo el flujo hacia arriba es más rápido en el centro que en la periferia.

Un gas poco común: inelasticidad, aglutinamiento, desplome

Existe una diferencia crítica entre los materiales granulares y los fluidos ordinarios: las interacciones entre granos son inherentemente inelásticas, por lo que se pierde un poco de energía a cada colisión. Esto implica que la mecánica estadística involucrada tiene una nueva gama de características. Cualquier analogía entre el comportamiento de los granulares y los fluidos, propiamente dicho, es netamente un fenómeno dinámico. Las ondas superficiales que se describieron en la sección anterior, por ejemplo, no son una respuesta lineal a un suministro externo de energía, sino la consecuencia de una transición altamente no–lineal e histerética entre dos estados

esencialmente sólidos. El comportamiento fluido es simple consecuencia de un suministro de energía por encima de un umbral; en cuanto el suministro cese, el flujo se detendrá por efecto de las colisiones inelásticas. Si dejamos caer una canica en un vidrio, aquella rebota durante largo tiempo. Un costalito que holgadamente contenga varias canicas idénticas a la anterior se para en seco tras impactarse con el vidrio. Esta marcada diferencia entre el comportamiento individual y el del conjunto es producto del gran número de rápidas colisiones inelásticas que se presentan entre canicas vecinas. De hecho, diversos dispositivos, como los martillos neumáticos, algunas armas de fuego y los vibradores de moldes para fundición, emplean esta característica para absorber energía.

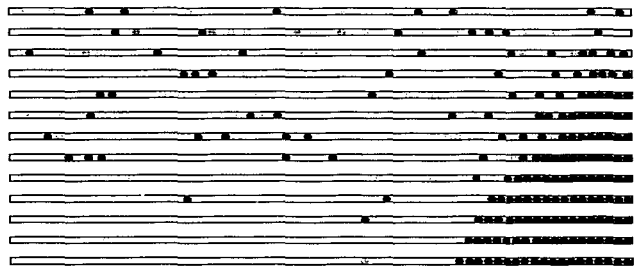


Figura 8. Las partículas comienzan con velocidades aleatorias a lo largo de la única dimensión libre. El extremo izquierdo es una fuente de energía, mientras que el extremo derecho es una pared fija. Los tonos más obscuros denotan velocidades más bajas. Conforme avanza el tiempo, se observa que las partículas se colapsan, quedando al final una sola con muy alta velocidad. Si ambas paredes estuvieran termalizadas, ¡se presentaría un colapso en el centro!

Dado que las colisiones son inelásticas, un "baño térmico" puede ser insuficiente para mantener el estado termodinámico del sistema. Si las partículas comienzan a aglutinarse se presentará un rompimiento del estado newtoniano del sistema debido a que estos conglomerados no están en posibilidades de "derretirse" de nuevo. En los últimos años se ha estudiado este efecto, descubriéndose un tipo especial llamado *colapso inelástico*. Sean McNamara y William Young[19] demostraron que la inelasticidad puede ocasionar que en un tiempo finito ocurra un número infinito de colisiones. En una dimensión (*Figura 8*), las partículas se colapsan en un núcleo que mantiene contacto permanente y sin movimiento relativo entre sus partículas; en dos y tres dimensiones, se producen densos conglomerados concatenados entre sí. Aún no se cuenta con una teoría que explique satisfactoriamente este tipo de fenómenos. Lo más notable es que los conglomerados que se forman no son amorfos sino más bien largas cadenas de granos. De hecho, el

parecido con los mapas de densidad del universo conocido es notable. Vale la pena meditar un momento y tomar en cuenta que el potencial de atracción gravitacional puede jugar un papel importante para mantener una densidad lo suficientemente alta como para que se formen cúmulos de partículas. Así pues, a gran escala, se podría considerar que las colisiones inelásticas son responsables, en parte, de la coagulación de gases que finalmente forma las estrellas y galaxias.

Después de todo, parece que Victor Hugo no propuso algo tan descabellado. Es más: es posible que el movimiento de los granos de arena sea de fundamental importancia para la creación, no sólo de mundos, sino incluso de galaxias—para la estructura y formación del paisaje astronómico. Aún queda mucho por aprender acerca de este conjunto de materiales metaforizados con la arena. Hay muchos detalles que, aún a sabiendas de que son relevantes, no se sabe cómo incluirlos en los modelos físicos. Como el mismo Victor Hugo mencionara en otra parte de su citado libro: "*en la arena...hay cierta finura que es pérfida*".

Referencias

[1] Jaeger, Nagel y Behringer, 1996a.

[2] Ogawa, 1978; Walton y Braun, 1986; Haff, 1986; Campbell, 1990; Ippolito *et al.*, 1995; Warr, Huntley y Jacques, 1995; Warr y Huntley, 1995.

[3] Coulomb, 1773; Faraday, 1831; Reynolds, 1885.

[4] Ennis *et al.*, 1994; Knowlton *et al.*, 1994.

[5] Liu *et al.*, 1995.

[6] Onoda y Liniger, 1990; Bideau y Dodds, 1991; Brooker *et al.*, 1992.

[7] Knight *et al.*, 1995.

[8] Ben-Naim *et al.*, 1996.

[9] Ogawa, 1978; Savage y Jeffrey, 1981; Jenkins y Savage, 1983; Haff, 1983 y 1986; Campbell, 1990.

[10] Reynolds, 1885.

[11] Jaeger, Nagel y Behringer, 1996b.

[12] Faraday, 1831.

[13] Knight *et al.*, 1993, 1995; Pak y Behringer, 1993, 1994; Lee, 1994; Pak *et al.*, 1995.

[14] Aoki *et al.*, 1996.

[15] Ehrichs *et al.*, 1995.

[16] Umbanhowar *et al.*, 1996.

[17] Nakagawa y Caprihan, 1994; Morris y Choo, 1996; Hill, Caprihan y Kakalios, 1997.

[18] Metcalfe *et al.*, 1995.

[19] McNamara y Young, 1994.

Métodos para Simulación Numérica de Flujos[1]

¿Por qué simular flujos?

Desde prácticamente el inicio de la década de 1980, se han vuelto cada vez más y más comunes las simulaciones de fenómenos físicos por medio de computadoras digitales. ¿A qué se debe esto? Una respuesta sencilla sería que cada vez las computadoras son más comunes, pero esta respuesta peca de ingenua y vaga. Una respuesta un poco más mesurada sería que, gracias a los desarrollos tecnológicos, las computadoras cada vez tienen más poder a un costo accesible para su aplicación en problemas de difícil resolución. Esto hace que se recurra a ellas como auxiliares en la solución de ecuaciones que no tienen resultado analítico directo. Pero la duda persiste: ¿por qué simular flujos (o cualquier otra cosa)? La respuesta a esta pregunta tiene mucho fondo.

Las ecuaciones de la mecánica estadística, a la cual se encuentran fuertemente ligados los problemas de flujo, sólo tienen solución analítica en contados casos. Esto nos obliga a recurrir, en caso de querer resolverlas para un problema determinado, a métodos aproximados que no siempre logran tener la precisión deseada. Sin embargo, en el caso de los fluidos probablemente ni siquiera quede claro por dónde empezar para generar una teoría aproximada que sea razonable. Mientras más complicado e interesante es un problema, lo más deseable que es obtener resultados exactos. A la vez, no se quiere introducir la duda de si en realidad el modelo empleado se asemeja o no a la realidad (a menos que estemos tratando de evaluar un modelo).

Es aquí donde las simulaciones numéricas toman un papel fundamental para obtener resultados esencialmente exactos en problemas que, aún haciendo aproximaciones, podrían ser intratables. En este sentido, las simulaciones por computadora sirven para probar teorías, al comparar

sus resultados con las mediciones experimentales. Las simulaciones por computadora también pueden dar "pistas" que auxilien al experimentalista a interpretar nuevos resultados. Este doble rol de las simulaciones numéricas hace que a veces se considere a la técnica numérica como *experimentación por computadora* (*Figura 9*).

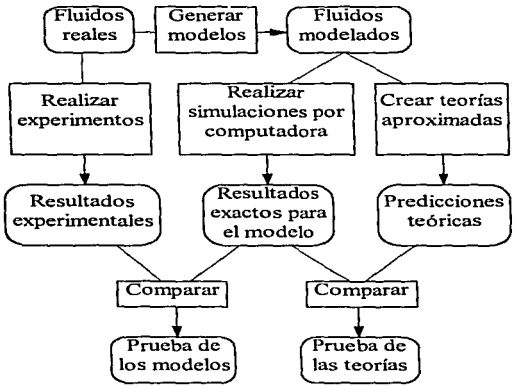


Figura 9. Relación entre experimentos, teoría y simulación por computadora.

Las simulaciones por computadora generalmente involucran sistemas con un número de partículas, N, del orden $10 \leq N \leq 10\,000$. El tamaño del sistema se ve limitado por la capacidad de almacenamiento de la computadora en la que se realice la simulación, pero sobre todo por la velocidad de ejecución. El tiempo de cómputo empleado en calcular las fuerzas típicamente es proporcional a N^2—con ciertos trucos se puede hacer que su dependencia sea $O(N)$—, pero definitivamente los sistemas pequeños son los más económicos. El problema es que, a menos que el sistema de interés sea pequeño por sí mismo, cualquier intento por confinar el sistema sólo logrará introducir efectos de frontera no deseables. Para "darle la vuelta" a este problema, se puede recurrir a fronteras periódicas. Se comienza por replicar la celda de simulación a lo largo de todo el espacio para formar una red infinita. Cada que una partícula se mueve a través de la celda original, su imagen periódica en cada una de las demás celdas se mueve exactamente de la misma manera. Así, cuando una partícula abandona la celda a través de una cara, una de sus

imágenes entra a través de la cara opuesta. No hay, por tanto, ni paredes ni partículas de frontera en ninguna celda.

Una alternativa viable para evitar el uso de fronteras periódicas consiste en referir un sistema n–dimensional a un espacio de orden n+1. Un sistema bidimensional se puede confinar en la superficie de una esfera, eliminando toda frontera. De igual forma se puede extender el concepto para embeber un sistema tridimensional en la superficie de una hiperesfera. Una complicación inevitable será la geometría no euclidiana involucrada con este tipo de topología. La simulación dentro de geometrías no euclidianas ayuda a reducir el tamaño de los sistemas al mínimo. Como no se cuenta con una periodicidad definida, estos sistemas son altamente compatibles con la simulación de líquidos. Su mayor debilidad se encuentra en la dificultad para simular, por este camino, sistemas con cambios de fase.

La primera simulación numérica realizada colocó los cimientos de lo que actualmente se conoce como método de *Monte Carlo*, llamado así por el papel preponderante que en él juega el azar (números aleatorios) para resolver un modelo determinado. Un método diferente se emplea para calcular las propiedades dinámicas de los sistemas de muchas partículas. A la solución acoplada de las ecuaciones clásicas del movimiento (ecuaciones de Newton) para un conjunto de partículas se le llama *Dinámica Molecular*. Con este método se conoce la velocidad de cada partícula entre una colisión y otra, por lo que se puede resolver el problema dinámico de manera exacta (en el supuesto de que se conozca de manera igualmente exacta el potencial de interacción para las colisiones). Si el potencial de interacción es, por ejemplo, de Lennard-Jones, las fuerzas cambian continuamente con el movimiento por lo que se requiere una aproximación paso a paso. En un caso así (con potenciales de interacción de largo alcance) resulta importante truncar los efectos a una cierta distancia, pues de lo contrario requeriríamos tomarlos en cuenta para todos y cada uno de los cálculos, complicando excesivamente la operabilidad del algoritmo. La forma en la que este truncamiento se lleve a cabo dependerá tanto del método de simulación empleado como de la topología sobre la cual se esté simulando, *i.e.* fronteras periódicas o fronteras no euclidianas.

Dinámica Molecular

Las ecuaciones clásicas del movimiento para un sistema de N partículas que interactúan por medio de un potencial $\mathcal{V}$ se pueden escribir de diversas maneras. Probablemente la forma más fundamental sea la lagrangiana:

$$\frac{d}{dt}(\partial \mathcal{L}/\partial \dot{q}_k) - (\partial \mathcal{L}/\partial q_k) = 0 .$$

El operador lagrangiano se define en función de la energía cinética y la potencial

$$\mathcal{L} = \mathcal{K} - \mathcal{V}$$

y es función de las coordenadas generalizadas y de sus derivadas temporales $\dot{q}_k$. Para un sistema de átomos en coordenadas cartesianas y con las definiciones usuales de $\mathcal{K}$ y $\mathcal{V}$, se obtiene

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i , \qquad \text{(enfoque lagrangiano)}$$

donde m_i es la masa de la partícula i y $\mathbf{f}_i = \nabla_{r_i}\mathcal{L} = -\nabla_{r_i}\mathcal{V}$ representa la fuerza total que actúa sobre el centro de masa de dicha partícula.

El momento generalizado p_k conjugado a q_k se define como

$$p_k = \partial \mathcal{L}/\partial \dot{q}_k .$$

Las forma hamiltoniana de las ecuaciones del movimiento es

$$\dot{q}_k = \partial \mathcal{H}/\partial p_k$$
$$\dot{p}_k = -\partial \mathcal{H}/\partial q_k$$

donde H es el operador hamiltoniano, que se define en función del momento y de las coordenadas generalizadas

$$\mathcal{H}(\mathbf{p},\mathbf{q}) = \sum_k \dot{q}_k p_k - \mathcal{L}(\mathbf{q},\dot{\mathbf{q}})$$

Para coordenadas cartesianas, las ecuaciones hamiltonianas se convierten en

$$\dot{\mathbf{r}}_i = \mathbf{p}_i/m_i$$
$$\dot{\mathbf{p}}_i = -\nabla_{r_i}\mathcal{V} = \mathbf{f}_i . \qquad \text{(enfoque hamiltoniano)}$$

En consecuencia, calcular la trayectoria de los centros de masa de N partículas implica resolver ya sea un sistema de $3N$ ecuaciones diferenciales de segundo orden (enfoque lagrangiano) o un sistema equivalente de $6N$ ecuaciones diferenciales de primer orden (enfoque hamiltoniano).

Para resolver estos sistemas se recurre típicamente a los métodos de diferencias finitas. La idea de fondo es, dadas las posiciones, velocidades y demás parámetros dinámicos en el tiempo t, tratar de calcular, con la suficiente precisión, las posiciones, velocidades y demás parámetros en un tiempo posterior $t + \delta t$. Se va resolviendo "paso a paso" las ecuaciones involucradas; el valor de δt dependerá en cierta forma del método empleado pero es, en general, mucho menor al tiempo típico que toma una partícula en recorrer su propio diámetro.

Si hiciéramos una pequeña lista de las características deseables para un algoritmo exitoso de simulación, podríamos enumerar:

(a) Debe ser rápido y consumir poca memoria.
(b) Debe permitir emplear pasos de tiempo grandes.
(c) Debe replicar lo más cercanamente posible la trayectoria clásica.
(d) Debe satisfacer las leyes de la conservación de la energía y ser reversible.
(e) Debe ser simple y fácil de programar.

Sin embargo, no todas estas características son importantes para la dinámica molecular. La velocidad del algoritmo es prácticamente irrelevante en contraste con el tardado cálculo de fuerzas. Es mucho más importante poder emplear una δt grande, para reducir al mínimo el número de integraciones requeridas para simular un periodo determinado de tiempo. Pero al hacer crecer el paso de tiempo, la trayectoria calculada se aleja cada vez más de la trayectoria clásica. ¿Cuán importantes son los incisos (c) y (d)?

La simulación numérica de un flujo implica evaluar un cierto sector, *i.e.* evaluar un sistema muy grande calculando valores en un sistema reducido. Esto significa que las trayectorias individuales no son muy importantes, pero sí lo es la conservación de la energía. Al aumentar el valor de δt se deteriora la conservación de la energía, por lo que un buen algoritmo será aquel que, manteniendo una buena conservación, permita un paso de tiempo grande. Así pues, se emplean pasos de tiempo menores para partículas ligeras, temperaturas altas y sistemas sujetos a potenciales que varían rápidamente.

Por último, un algoritmo debe ser simple. Esto significa que debe implicar almacenar pocos valores de las posiciones, velocidades, etc., y que debe ser fácil de programar. No vale la pena desperdiciar esfuerzos en programar un algoritmo complejo. Es mejor optimar el cálculo de fuerzas, pues así desperdiciaremos menos tiempo que el que ganaríamos con un algoritmo más ágil.

Algoritmo de Gear de predicción–corrección

Si la trayectoria clásica es continua, las posiciones, velocidades y demás parámetros pueden estimarse en un tiempo $t+\delta t$ por expansión simple de Taylor en torno a t:

$$\mathbf{r}^p(t+\delta t) = \mathbf{r}(t) + \delta t\mathbf{v}(t) + \tfrac{1}{2}\delta t^2\mathbf{a}(t) + \tfrac{1}{6}\delta t^3\mathbf{b}(t) + \ldots$$
$$\mathbf{v}^p(t+\delta t) = \mathbf{v}(t) + \delta t\mathbf{a}(t) + \tfrac{1}{2}\delta t^2\mathbf{b}(t) + \ldots$$
$$\mathbf{a}^p(t+\delta t) = \mathbf{a}(t) + \delta t\mathbf{b}(t) + \ldots$$
$$\mathbf{b}^p(t+\delta t) = \mathbf{b}(t) + \ldots \quad .$$

El superíndice indica que estos valores son producto de una predicción y que en breve serán corregidos. Los vectores **r**, **v**, **a** y **b** representan respectivamente las posiciones, velocidades, aceleraciones y la tercera derivada temporal de la posición. Sin embargo, estas ecuaciones no generan trayectorias correctas conforme el tiempo avanza, pues no incluyen a las ecuaciones del movimiento. Estas últimas entran en juego en el paso de corrección. A partir de las nuevas posiciones $\mathbf{r}^p$ podemos calcular las fuerzas (y por ende las aceleraciones) en el tiempo $t+\delta t$.

Si comparamos el valor obtenido para la aceleración con el valor producto de la predicción, obtenemos un estimado del error en el paso de predicción

$$\Delta\mathbf{a}(t+\delta t) = \mathbf{a}^c(t+\delta t) - \mathbf{a}^p(t+\delta t)$$

que podemos introducir a un paso de corrección tal que

$$\mathbf{r}^c(t+\delta t) = \mathbf{r}^p(t+\delta t) + c_0\Delta\mathbf{a}(t+\delta t)$$
$$\mathbf{v}^c(t+\delta t) = \mathbf{v}^p(t+\delta t) + c_1\Delta\mathbf{a}(t+\delta t)$$
$$\mathbf{a}^c(t+\delta t) = \mathbf{a}^p(t+\delta t) + c_2\Delta\mathbf{a}(t+\delta t)$$
$$\mathbf{b}^c(t+\delta t) = \mathbf{b}^p(t+\delta t) + c_3\Delta\mathbf{a}(t+\delta t).$$

La idea es que $\mathbf{r}^c(t+\delta t)$, etc., son mejores aproximaciones a las posiciones, velocidades, etc. reales. Gear propuso entre 1966 y 1971 el conjunto 'óptimo' de coeficientes c_0, c_1, c_2, c_3... para lograr la máxima estabilidad y precisión de las trayectorias.

El paso de corrección puede iterarse para obtener nuevas aceleraciones corregidas a partir de las posiciones recién corregidas. Esto es la clave, en muchos casos, para obtener un resultado preciso. La predicción provee un estimado inicial de la solución—que en principio no tiene que ser muy bueno—y los pasos sucesivos de corrección convergen rápidamente a la solución correcta. Sin embargo, es precisamente el cálculo de fuerzas lo que más tiempo de cómputo consume

en dinámica molecular. Como este cálculo está implícito en el paso de corrección, un número grande de iteraciones resultaría muy costoso. Es por eso que se requiere de un paso de predicción muy preciso, para poder disminuir el número de iteraciones de corrección a una o a lo sumo dos.

Algoritmo de Verlet

Éste es probablemente el método más difundido para integrar las ecuaciones de movimiento. Este método se basa en las posiciones $\mathbf{r}(t)$, las aceleraciones $\mathbf{a}(t)$ y las posiciones $\mathbf{r}(t-\delta t)$ para resolver de manera directa las ecuaciones de segundo orden del enfoque lagrangiano. La ecuación para actualizar las posiciones,

$$\mathbf{r}(t+\delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\delta t) + \delta t^2 \mathbf{a}(t),$$

tiene varias peculiaridades. Para empezar, la velocidad no aparece en ningún lado, pues se le eliminó al sumar las expansiones de Taylor en torno a $\mathbf{r}(t)$:

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t \mathbf{v}(t) + \tfrac{1}{2}\delta t^2 \mathbf{a}(t) + \ldots$$
$$\mathbf{r}(t-\delta t) = \mathbf{r}(t) - \delta t \mathbf{v}(t) + \tfrac{1}{2}\delta t^2 \mathbf{a}(t) - \ldots \ .$$

Así pues, no necesitamos de las velocidades para calcular las trayectorias, pero sí para calcular la energía cinética (y por tanto, la energía total). La velocidad se obtiene de la fórmula

$$\mathbf{v}(t) = \frac{\mathbf{r}(t+\delta t) - \mathbf{r}(t-\delta t)}{2\delta t}.$$

El error asociado a la posición es del orden de δt^4, mientras que el asociado a la velocidad es del orden de δt^2. Una pequeña inconveniencia radica en que necesitamos conocer $\mathbf{r}(t+\delta t)$ para calcular $\mathbf{v}(t)$. Por otra parte, el algoritmo de Verlet está propiamente centrado, *i.e.* $\mathbf{r}(t-\delta t)$ y $\mathbf{r}(t+\delta t)$ juegan papeles simétricos, por lo que es reversible en el tiempo. Además, el paso de tiempo se da de un solo golpe, en oposición al esquema de predicción–corrección ya descrito, que requiere de dos etapas por cada paso de tiempo. En cuanto a las aceleraciones, éstas se obtienen de la rutina de fuerzas.

Para este algoritmo se requiere almacenar $9N$ valores, lo cual lo hace tanto compacto como fácil de programar. Es exactamente reversible en el tiempo y, dadas fuerzas conservativas, la conservación del momento lineal está garantizada. El algoritmo es muy estable aún para pasos de

tiempo largos. En contra de este algoritmo se debe mencionar que el manejo de las velocidades es más bien extraño, lo cual podría introducir cierto grado de imprecisión numérica.

Para subsanar esta deficiencia existen varias modificaciones al algoritmo básico de Verlet. Uno de estos algoritmos modificados es el método del *salto de rana*. Su nombre resulta evidente cuando se le representa esquemáticamente (*Figura 10*).

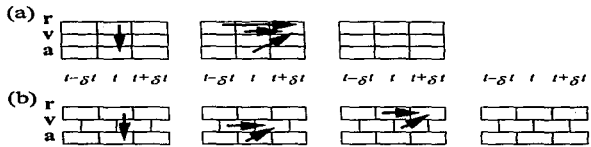


Figura 10. Dos formas comunes del algoritmo de Verlet. La secuencia (a) muestra el método original de Verlet. La secuencia (b) corresponde al método de *salto de rana*. Los valores almacenados se representan por las celdas sombreadas.

Las ecuaciones involucradas

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \delta t\, \mathbf{v}\left(t+\tfrac{1}{2}\delta t\right)$$
$$\mathbf{v}\left(t+\tfrac{1}{2}\delta t\right) = \mathbf{v}\left(t-\tfrac{1}{2}\delta t\right) + \delta t\, \mathbf{a}(t)$$

implican almacenar las posiciones $\mathbf{r}(t)$ y aceleraciones $\mathbf{a}(t)$ actuales y las velocidades $\mathbf{v}(t-\frac{1}{2}\delta t)$. Las velocidades y posiciones se van saltando entre sí, apoyadas en el cálculo de fuerzas para determinar las aceleraciones.

Para poder calcular la energía total del sistema en el tiempo t, es necesario calcular la velocidad actual

$$v(t) = \tfrac{1}{2}\left(v\left(t+\tfrac{1}{2}\delta t\right) + v\left(t-\tfrac{1}{2}\delta t\right)\right),$$

ya que el cálculo de la energía cinética requiere del valor de la velocidad.

Este método presenta la ventaja de que las velocidades aparecen de manera explícita (aunque no aparezcan en el tiempo t). Esto facilita ajustar la energía de la simulación, normalizando las velocidades para mantener la energía cinética total constante.

Método de Monte Carlo

El método de Monte Carlo fue desarrollado por von Neumann, Ulam y Metropolis a fines de la Segunda Guerra Mundial para estudiar la difusión de neutrones en un material fisionable. Se escogió el nombre de 'Monte Carlo', acuñado por Metropolis, debido a que este método hace un uso exhaustivo de los números aleatorios.

Diversos científicos (Lord Kelvin, entre ellos) habían empleado previamente el método de muestreo experimental para resolver problemas. La contribución de von Neumann y Ulam consistió en darse cuenta que diversos problemas matemáticos se pueden resolver por muestreo estocástico dentro de un problema probabilístico análogo. El muestreo se lleva a cabo calculando números aleatorios y luego sometiéndolos a un conjunto limitado de operaciones aritméticas y lógicas. Es por esto natural que con el advenimiento de las computadoras este método tomara fuerza, estando fuertemente ligado al desarrollo tecnológico de las mismas.

Claro está que algunas personas no pueden esperar a que la tecnología evolucione. Hay un hermoso teorema en matemáticas, descubierto por Buffon hace unos dos siglos. Este teorema dice que si aventamos al azar una aguja de longitud l sobre un conjunto de paralelas equiespaciadas a una distancia d (siendo $d > l$), la probabilidad de que la aguja atraviese una línea es igual a $2l/\pi d$. Basándose en este teorema, el matemático italiano Lazzerini estimó en 1901 que el valor de π es 3.1415929, dejando caer una aguja 3407 veces. Por un método semejante (pero empleando computadoras digitales), se ha calculado en la actualidad muchas más cifras de π.

Dos cosas resultan importantes para el éxito de una simulación de Monte Carlo, a saber, un buen generador de números aleatorios y los criterios correctos para equiparar el problema físico con el problema probabilístico. El método de Monte Carlo no se adapta muy bien a la solución de sistemas granulares, por lo que no profundizaré más en su análisis.

En el programa empleado para las simulaciones numéricas de esta tesis, se utiliza un método simplificado de Monte Carlo para calcular las fracciones de empaquetamiento del sistema. En dicho método, se determinan cien mil posiciones aleatorias dentro de la celda de simulación. Con base en la distancia al vecino más cercano a cada una de las posiciones determinadas, se estima la fracción total de empaquetamiento.

Referencias

[1] Allen y Tildesley, 1987.

Descripción del Algoritmo Empleado

El método numérico empleado es una extensión a tres dimensiones de los modelos de fuerza de contacto bidimensional y de las ecuaciones integrales de Walton y Braun[1] (modelo WB), que están basados en algoritmos de dinámica molecular. El código empleado—con todo y las modificaciones realizadas—y su diagrama de flujo se presentan en el Apéndice A.

La integración explícita de las ecuaciones de movimiento se logra por medio del método de *salto de rana* para cada uno de los seis grados de libertad.[2] Las posiciones y fuerzas se conocen al final de cada paso de tiempo[a] mientras que las velocidades se conocen a la mitad del paso.

Cálculo de posiciones, orientaciones y velocidades

Las ecuaciones de Newton se expresan como ecuaciones diferenciales de primer orden en cada dimensión espacial para cada partícula:

$$\dot{v}_\alpha = g_\alpha + \frac{F_\alpha}{m}, \quad \alpha = x, y, z \ ;$$
$$\dot{r}_\alpha = v_\alpha \ , \qquad \alpha = x, y, z \ .$$

Las derivadas centrales serán:

$$v_\alpha^{n+\frac{1}{2}} = v_\alpha^{n+\frac{1}{2}} + \Delta t\left(g_\alpha + \frac{F_\alpha^n}{m}\right) \qquad \alpha = x, y, z \ ;$$

$$r_\alpha^{n+1} = r_\alpha^n + \Delta t \, v_\alpha^{n+\frac{1}{2}} \ , \qquad \alpha = x, y, z$$

[a] En adelante, se le llamará paso de tiempo (o simplemente paso) a cada intervalo de incremento de la variable temporal.

siendo el superíndice el número del paso, r la posición, v la velocidad, F las fuerza de contacto o de cuerpo, m la masa y g la aceleración gravitacional.

Las derivadas temporales de la velocidad angular están definidas en el marco principal de referencia[a] por las ecuaciones de movimiento de Euler,

$$\dot{\omega}_x = \left[N_{px} + \omega_y \omega_z \left(I_{py} - I_{pz}\right)\right] / I_{px}$$
$$\dot{\omega}_y = \left[N_{py} + \omega_z \omega_x \left(I_{pz} - I_{px}\right)\right] / I_{py}$$
$$\dot{\omega}_z = \left[N_{pz} + \omega_x \omega_y \left(I_{px} - I_{py}\right)\right] / I_{pz}$$

donde $\mathbf{I}_p$ es el tensor (diagonal) de momento de inercia en el marco principal de referencia y $\mathbf{N}_p$ es el vector de torque en el marco principal de referencia.

Debido a que aparecen productos de velocidades angulares en el miembro derecho de las ecuaciones de Euler, se emplea un algoritmo del tipo predicción–corrección para integrar las derivadas de las velocidades angulares. Los torques se conocen *en* cada paso de tiempo y las velocidades angulares se conocen a la *mitad* de cada paso. Primero, se estima la velocidad angular en el paso actual suponiendo que la aceleración angular se mantiene constante por medio paso de tiempo más

$$\omega_\alpha'^{\,n} = \omega_\alpha^{n-\frac{1}{2}} + \frac{\Delta\omega_\alpha^{n-1}}{2}, \quad \alpha = x, y, z .$$

Esta velocidad angular *extrapolada* se emplea, junto con el torque actual, para hacer una primera *predicción* de las aceleraciones angulares en el paso actual

$$\Delta\omega_x'^{\,n} = \left[N_{px}^n + \omega_y'^{\,n} \omega_z'^{\,n} \left(I_{py} - I_{pz}\right)\right] \frac{\Delta t}{I_{px}}$$
$$\Delta\omega_y'^{\,n} = \left[N_{py}^n + \omega_z'^{\,n} \omega_x'^{\,n} \left(I_{pz} - I_{px}\right)\right] \frac{\Delta t}{I_{py}}$$
$$\Delta\omega_z'^{\,n} = \left[N_{pz}^n + \omega_x'^{\,n} \omega_y'^{\,n} \left(I_{px} - I_{py}\right)\right] \frac{\Delta t}{I_{pz}} .$$

Las aceleraciones angulares predecidas se emplean para estimar de manera más exacta la velocidad angular en el paso actual

[a] Marco de referencia relativo a cada partícula (marco principal o de cuerpo, también conocido como euleriano), en contraposición al de todo el sistema (marco espacial o absoluto, también conocido como lagrangiano).

$$\omega_\alpha^n = \omega_\alpha^{n-\frac{1}{2}} + \frac{\Delta\omega_\alpha'^n}{2}, \quad \alpha = x, y, z.$$

Los valores corregidos para las derivadas serán entonces

$$\Delta\omega_x^n = \left[N_{px}^n + \omega_y^n \omega_z^n \left(I_{py} - I_{pz}\right)\right]\frac{\Delta t}{I_{px}}$$

$$\Delta\omega_y^n = \left[N_{py}^n + \omega_z^n \omega_x^n \left(I_{pz} - I_{px}\right)\right]\frac{\Delta t}{I_{py}}$$

$$\Delta\omega_z^n = \left[N_{pz}^n + \omega_x^n \omega_y^n \left(I_{px} - I_{py}\right)\right]\frac{\Delta t}{I_{pz}}.$$

En este punto se pueden emplear los valores *corregidos* para directamente actualizar las velocidades angulares a la mitad del siguiente paso,

$$\omega_\alpha^{n+\frac{1}{2}} = \omega_\alpha^{n-\frac{1}{2}} + \frac{\Delta\omega_\alpha^n}{2}, \quad \alpha = x, y, z;$$

o bien seguir iterando en las seis últimas ecuaciones hasta alcanzar un criterio de convergencia basado en el cambio de $\Delta\omega_\alpha^n$ de una iteración a la siguiente o hasta completar un número predeterminado de iteraciones.

La orientación de cada partícula se calcula por medio de una adaptación de los cuaterniones definidos por Evans.[3] Para emplear las ecuaciones de movimiento de Euler e integrar los cuaterniones, los torques se deben especificar en el marco principal de referencia, mientras que la detección de los contactos y el cálculo de fuerzas se realizan en el marco espacial de referencia. La matriz de rotación para transformar del marco *espacial* al marco *de cuerpo* está dada por:

$$\mathbf{A} = \begin{bmatrix} -q_1^2 + q_2^2 - q_3^2 + q_4^2 & -2(q_1q_2 - q_3q_4) & 2(q_2q_3 + q_1q_4) \\ -2(q_1q_2 + q_3q_4) & q_1^2 - q_2^2 - q_3^2 + q_4^2 & -2(q_1q_3 - q_2q_4) \\ 2(q_2q_3 - q_1q_4) & -2(q_1q_3 + q_2q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}$$

siendo

$$q_1 = \operatorname{sen}\tfrac{\theta}{2}\operatorname{sen}\left(\tfrac{\psi-\phi}{2}\right), \quad q_2 = \operatorname{sen}\tfrac{\theta}{2}\cos\left(\tfrac{\psi-\phi}{2}\right), \quad q_3 = \cos\tfrac{\theta}{2}\operatorname{sen}\left(\tfrac{\psi+\phi}{2}\right), \quad q_4 = \cos\tfrac{\theta}{2}\cos\left(\tfrac{\psi+\phi}{2}\right)$$

y ϕ, θ, ψ los ángulos de Euler para las rotaciones sucesivas en torno a los ejes Z, X' y Z'.[4]

Las derivadas temporales de los cuaterniones se pueden expresar en función de los mismos cuaterniones y de las velocidades angulares

$$\dot{q}_1 = \tfrac{1}{2}\left(-q_3\omega_x - q_4\omega_y + q_2\omega_z\right)$$
$$\dot{q}_2 = \tfrac{1}{2}\left(q_4\omega_x - q_3\omega_y - q_1\omega_z\right)$$
$$\dot{q}_3 = \tfrac{1}{2}\left(q_1\omega_x + q_2\omega_y + q_4\omega_z\right)$$
$$\dot{q}_4 = \tfrac{1}{2}\left(-q_2\omega_x + q_1\omega_y - q_3\omega_z\right)$$

Al ser independientes sólo tres de estas ecuaciones, se requiere de la relación de normalización

$$\sum_{i=1}^{4} q_i^2 = 1$$

para que el sistema quede definido.

Las derivadas centrales equivalentes a las derivadas temporales de los cuaterniones se pueden resolver explícitamente para conocer los valores de los cuaterniones en el nuevo paso de tiempo, empleando los valores anteriores de los cuaterniones y las velocidades angulares a la mitad del paso. Las diferencias finitas de estas ecuaciones tienen la forma

$$q_1^{n+1} = q_1^n + \tfrac{\Delta t}{4}\left[-\left(q_3^{n+1} + q_3^n\right)\omega_x^{n+\frac{1}{2}} - \left(q_4^{n+1} + q_4^n\right)\omega_y^{n+\frac{1}{2}} + \left(q_2^{n+1} + q_2^n\right)\omega_z^{n+\frac{1}{2}}\right]$$
$$q_2^{n+1} = q_2^n + \tfrac{\Delta t}{4}\left[\left(q_4^{n+1} + q_4^n\right)\omega_x^{n+\frac{1}{2}} - \left(q_3^{n+1} + q_3^n\right)\omega_y^{n+\frac{1}{2}} - \left(q_1^{n+1} + q_1^n\right)\omega_z^{n+\frac{1}{2}}\right]$$
$$q_3^{n+1} = q_3^n + \tfrac{\Delta t}{4}\left[\left(q_1^{n+1} + q_1^n\right)\omega_x^{n+\frac{1}{2}} + \left(q_2^{n+1} + q_2^n\right)\omega_y^{n+\frac{1}{2}} + \left(q_4^{n+1} + q_4^n\right)\omega_z^{n+\frac{1}{2}}\right]$$
$$q_4^{n+1} = q_4^n + \tfrac{\Delta t}{4}\left[-\left(q_2^{n+1} + q_2^n\right)\omega_x^{n+\frac{1}{2}} + \left(q_1^{n+1} + q_1^n\right)\omega_y^{n+\frac{1}{2}} - \left(q_3^{n+1} + q_3^n\right)\omega_z^{n+\frac{1}{2}}\right].$$

Reacomodando las ecuaciones, tenemos

$$\begin{aligned}
q_1^{n+1} - \beta_z q_2^{n+1} + \beta_x q_3^{n+1} + \beta_y q_4^{n+1} &= q_1^n + \beta_z q_2^n - \beta_x q_3^n - \beta_y q_4^n \\
\beta_z q_1^{n+1} + q_2^{n+1} + \beta_y q_3^{n+1} - \beta_x q_4^{n+1} &= -\beta_z q_1^n + q_2^n - \beta_y q_3^n + \beta_x q_4^n \\
-\beta_x q_1^{n+1} - \beta_y q_2^{n+1} + q_3^{n+1} - \beta_z q_4^{n+1} &= \beta_x q_1^n + \beta_y q_2^n + q_3^n + \beta_z q_4^n \\
-\beta_y q_1^{n+1} + \beta_x q_2^{n+1} + \beta_z q_3^{n+1} + q_4^{n+1} &= \beta_y q_1^n - \beta_x q_2^n - \beta_z q_3^n + q_4^n
\end{aligned}$$

donde

$$\beta_x = \tfrac{\Delta t}{4}\omega_x^{n+\frac{1}{2}}, \quad \beta_y = \tfrac{\Delta t}{4}\omega_y^{n+\frac{1}{2}}, \quad \beta_z = \tfrac{\Delta t}{4}\omega_z^{n+\frac{1}{2}}.$$

Sea **B** la matriz de coeficientes de los miembros izquierdos de las ecuaciones anteriores y C_i, i=1, 2, 3, 4 los miembros derechos de las cuatro ecuaciones:

$$\mathbf{B}=\begin{bmatrix} 1 & -\beta_z & \beta_x & \beta_y \\ \beta_z & 1 & \beta_y & -\beta_x \\ -\beta_x & -\beta_y & 1 & -\beta_z \\ -\beta_y & \beta_x & \beta_z & 1 \end{bmatrix}$$

$$C_1 = q_1^n + \beta_z q_2^n - \beta_x q_3^n - \beta_y q_4^n$$
$$C_2 = -\beta_z q_1^n + q_2^n - \beta_y q_3^n + \beta_x q_4^n$$
$$C_3 = \beta_x q_1^n + \beta_y q_2^n + q_3^n + \beta_z q_4^n$$
$$C_4 = \beta_y q_1^n - \beta_x q_2^n - \beta_z q_3^n + q_4^n$$

Entonces

$$\det \mathbf{B} = 1 + 2\beta_x^2 + 2\beta_y^2 + 2\beta_z^2 + 2\beta_x^2\beta_y^2 + 2\beta_y^2\beta_z^2 + 2\beta_x^2\beta_z^2 + \beta_x^4 + \beta_y^4 + \beta_z^4$$

y

$$q_1^{n+1} = \left(C_1 + C_2\beta_z - C_3\beta_x - C_4\beta_y\right)\left(1 + \beta_x^2 + \beta_y^2 + \beta_z^2\right)/\det \mathbf{B}$$
$$q_2^{n+1} = \left(-C_1\beta_z + C_2 - C_3\beta_y + C_4\beta_x\right)\left(1 + \beta_x^2 + \beta_y^2 + \beta_z^2\right)/\det \mathbf{B}$$
$$q_3^{n+1} = \left(C_1\beta_x + C_2\beta_y + C_3 + C_4\beta_z\right)\left(1 + \beta_x^2 + \beta_y^2 + \beta_z^2\right)/\det \mathbf{B}$$
$$q_4^{n+1} = \left(C_1\beta_y - C_2\beta_x - C_3\beta_z + C_4\right)\left(1 + \beta_x^2 + \beta_y^2 + \beta_z^2\right)/\det \mathbf{B}\,.$$

Estas expresiones explícitas para los nuevos valores de q_i están centradas en el tiempo, por lo que se ahorran los pasos de predicción–corrección planteados por Allen y Tildesley.[5]

Aún cuando los cuaterniones satisfacen por definición la relación de normalización, es preferible asegurarse de que los errores de redondeo no hagan que la normalización falle. Es por ello que un factor de escala

$$f = \left[\sum_{i=1}^{4}\left(q_i^{n+1}\right)^2\right]^{-\frac{1}{2}}$$

se emplea para garantizar la normalización para cada partícula,

$$q_i'^{\,n+1} = f q_i^{n+1}, \quad i = 1, 2, 3, 4;$$

después de cada paso de integración. Todo este procedimiento de cálculo es directamente programable en lenguaje Fortran.

Cálculo de fuerzas

Se define una "piel" radial para la búsqueda de vecinos cercanos a cada partícula. Estos vecinos se almacenan en un conjunto de n_p+1 listas ligadas y entrelazadas entre ellas (una lista para cada una de las n_p partículas y una lista de direcciones de memoria previamente liberadas que pueden reutilizarse para nuevos vecinos). Cada que una pareja de vecinos se aleja más allá de la distancia de búsqueda, se elimina su registro de la lista de vecinos a la que pertenece y se agrega a la lista de direcciones vacías. Todo el conjunto de listas ligadas se actualiza a intervalos irregulares, cada que el desplazamiento de alguna partícula excede al radio de búsqueda.

Las fuerzas de contacto (y torques) se calculan para cada par de vecinos cercanos que en efecto presente una superposición. El modelo de desplazamiento de fuerzas incluye una histéresis dependiente de la posición tanto para la fuerza normal como para la tangencial (de fricción). La fuerza de contacto en la dirección normal se modela con una trayectoria lineal de carga (con pendiente K_1) acoplada a una trayectoria de descarga un poco más rígida (con pendiente K_2) de tal manera que una colisión binaria, aislada y sin fricción presenta un valor constante para el coeficiente de restitución $e = \sqrt{K_1/K_2}$, $K_2 > K_1$. La fuerza normal (modelo WB) está dada por

$$F_N = \begin{cases} K_1\alpha\,, & \text{para la carga,} \\ K_2(\alpha - \alpha_0)\,, & \text{para la descarga,} \end{cases}$$

siendo α el "traslape" de las partículas en contacto y con α_0 representando el "traslape" relativo, debido a la deformación inelástica de las superficies, cuando la fuerza en descarga se vuelve cero. El área entre las trayectorias de carga y descarga representa la energía perdida durante la deformación plástica. Para representar mejor las deformaciones permanentes, puede ser útil el caso en el que la pendiente de descarga, K_2, crezca linealmente con la magnitud de la fuerza máxima que se haya presentado en toda la historia de colisiones del sistema, *e.g.* $K_2 = K_0 + S\,F_{\text{máx}}$, siendo S un parámetro empírico determinado experimentalmente. Asimismo, podemos elevar los términos de desplazamiento α y $\alpha - \alpha_0$ a alguna potencia, digamos 3/2, para hacer el modelo equivalente al modelo propuesto por Hertz en 1881.[6]

Al ser las partículas esféricas, sólo se requiere de la magnitud y de la dirección de la velocidad angular para calcular los desplazamientos infinitesimales que presenta la superficie a cada paso de tiempo, *i.e.* la información necesaria para calcular los cambios en la fuerza tangencial.

La fuerza tangencial se acumula de manera no lineal con los desplazamientos tangenciales finitos que se presentan después de iniciado el contacto. Se determinan por separado los desplazamientos tangenciales *paralelo* y *perpendicular* a la fuerza de fricción instantánea. Se combina vectorialmente a ambos y se compara la suma obtenida con el límite total de fricción, μF_N, siendo μ el coeficiente de fricción. El comportamiento resultante es muy parecido al análisis hecho por Mindlin[7] para contactos para esferas friccionantes elásticas.

La rigidez tangencial efectiva en la dirección *paralela* a la fuerza de fricción existente está dada por

$$K_T = \begin{cases} K_0\left(1-\dfrac{T-T^*}{\mu F_N - T^*}\right)^{\gamma}, & \text{para } T \text{ creciente,} \\ K_0\left(1-\dfrac{T^*-T}{\mu F_N + T^*}\right)^{\gamma}, & \text{para } T \text{ decreciente,} \end{cases}$$

siendo K_0 la rigidez tangencial inicial; T, la magnitud actual de la fuerza tangencial; T^* comienza valiendo cero, pero toma el valor de la fuerza tangencial total T cada que su magnitud cambia de creciente a decreciente, o viceversa; γ es un parámetro que, en caso de valer 1/3, hace que el modelo se asemeje al modelo de Mindlin para contactos de esferas elásticas con coeficiente de fricción μ. Para una mejor analogía con el comportamiento de esferas sujetas a deformación plástica, un valor adecuado de γ es de 1 o 2.

Este modelo de fuerza implica cierto manejo del álgebra vectorial, debido a que la orientación de la superficie de contacto cambia continuamente durante una colisión típica. Para fines del modelo se supone que estos desplazamientos son relativamente pequeños de un paso de tiempo al siguiente. Sea $\hat{\mathbf{k}}_{ij}$ el vector unitario instantáneo del centro de la esfera I al centro de la esfera J,

$$\hat{\mathbf{k}}_{ij} = \frac{\mathbf{r}_j - \mathbf{r}_i}{\left|\mathbf{r}_j - \mathbf{r}_i\right|}.$$

Este vector es también el vector normal unitario en el punto de contacto. Proyectando la fuerza tangencial $\mathbf{T}_{\text{prev}}$ del paso previo al plano tangente actual

$$\mathbf{T}_0 = \mathbf{k}_{ij} \times \mathbf{T}_{\text{prev}} \times \mathbf{k}_{ij} = \mathbf{T}_{\text{prev}} - \mathbf{k}_{ij}\left(\mathbf{k}_{ij} \cdot \mathbf{T}_{\text{prev}}\right)$$

y normalizándola a la magnitud anterior, obtenemos un vector "inicial" $\mathbf{T}$ al cual agregar los cambios que se hayan presentado durante el último paso de tiempo

$$\mathbf{T} = \left|T_{\text{prev}} / T_0\right| \mathbf{T}_0 .$$

Conviene también calcular el vector unitario $\hat{\mathbf{t}} = \mathbf{T}/|T|$ en la dirección de esta fricción "inicial".

El desplazamiento superficial relativo durante el último paso de tiempo se proyecta sobre el plano tangente de contacto

$$\begin{aligned}\Delta\mathbf{s}^{n-\frac{1}{2}} &= \left[\hat{\mathbf{k}}_{ij} \times \left(\mathbf{v}_j^{n-\frac{1}{2}} - \mathbf{v}_i^{n-\frac{1}{2}}\right) \times \hat{\mathbf{k}}_{ij} + \rho_i\left(\vec{\omega}_i^{n-\frac{1}{2}} \times \hat{\mathbf{k}}_{ij}\right) + \rho_j\left(\vec{\omega}_j^{n-\frac{1}{2}} \times \hat{\mathbf{k}}_{ij}\right)\right]\Delta t \\ &\approx \Delta\mathbf{r}_{ij} - \hat{\mathbf{k}}_{ij}\left(\hat{\mathbf{k}}_{ij} \cdot \Delta\mathbf{r}_{ij}\right) + \left[\rho_i\left(\vec{\omega}_i^{n-\frac{1}{2}} \times \hat{\mathbf{k}}_{ij}\right) + \rho_j\left(\vec{\omega}_j^{n-\frac{1}{2}} \times \hat{\mathbf{k}}_{ij}\right)\right]\Delta t ,\end{aligned}$$

donde $\Delta\mathbf{r}_{ij} = \mathbf{r}_{ij}^{n} - \mathbf{r}_{ij}^{n-1}$ es el cambio del vector de posición relativa durante el paso de tiempo Δt. Los subíndices i y j de la velocidad $\mathbf{v}$, la velocidad angular $\vec{\omega}$ y el radio ρ indican, respectivamente, las esferas I o J.

Los desplazamientos paralelo y perpendicular respecto a la fuerza de fricción previa son

$$\Delta\mathbf{s}_{\parallel} = \left(\Delta\mathbf{s}^{n-\frac{1}{2}} \cdot \hat{\mathbf{t}}\right)\hat{\mathbf{t}},$$

$$\Delta\mathbf{s}_{\perp} = \Delta\mathbf{s}^{n-\frac{1}{2}} - \Delta\mathbf{s}_{\parallel} .$$

Si el valor de la fuerza normal F_N cambia de un paso de tiempo al siguiente, se escala el valor de T^* proporcionalmente al cambio de la fuerza normal

$$T^{*\prime} = T^* \left|\frac{F_N^n}{F_N^{n-1}}\right| .$$

La rigidez tangencial incremental efectiva K_T se calcula de la ecuación correspondiente, según sea T creciente o decreciente, substituyendo con $T^{*\prime}$ a T^*.

Se calcula un nuevo valor

$$\mathbf{T}_{\parallel} = \mathbf{T} + K_T \Delta\mathbf{s}_{\parallel}$$

para la componente de la fuerza de fricción paralela a la fuerza de fricción previa. Si simultáneamente se satisface que $\Delta\mathbf{s}^{n-\frac{1}{2}} \cdot \hat{\mathbf{t}} < 0$ y $T + \left(\Delta\mathbf{s}^{n-\frac{1}{2}} \cdot \hat{\mathbf{t}}\right)K_T < 0$, implica que se ha invertido la dirección de $\mathbf{T}_{\parallel}$; en el modelo, se invierte el signo del punto de cambio "recordado", *i.e.* se cambia T^* por $-T^*$, de tal forma que el valor de la rigidez tangencial varía suavemente.

La fuerza tangencial perpendicular a la fricción será simplemente

$$\mathbf{T}_{\perp} = K_0 \Delta \mathbf{s}_{\perp},$$

pues no se supone que haya esfuerzos previos en la dirección perpendicular a la fricción.

La nueva fuerza tangente será, tentativamente,

$$\mathbf{T}' = \mathbf{T}_{\parallel} + \mathbf{T}_{\perp}.$$

Se compara el valor con el límite de fricción, μF_N. En caso de que lo exceda, se le escala para hacer que la fricción sea igual al citado límite. El desarrollo de este modelo de fuerzas en Fortran es directo y sin ambigüedades.

Modificaciones al Código Original

Para poder plantear el problema de flujo dentro del silo, realicé modificaciones importantes al programa `3dshear` de O. Walton y R. Braun. Basándome en la versión 2.05b de dicho programa, introduje los conceptos de planos finitos arbitrarios y de ejes de simetría. Ambos conceptos son fundamentales para simular una geometría del tipo de la del silo hexagonal.

Para definir los planos arbitrarios, había primero que encontrar la manera de representar un plano en el modelo descrito. Una manera sencilla de hacer esto consiste en apoyarnos en el concepto de partícula para definir el plano. Ya se mencionó que los cálculos de interacción se realizan sólo para las parejas de vecinos cercanos que se estén superponiendo. En el caso de dos esferas I y J, la distancia entre ellas estará dada por el vector $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$, o en componentes

$$\delta r_x = r_{jx} - r_{ix},$$
$$\delta r_y = r_{jy} - r_{iy},$$
$$\delta r_z = r_{jz} - r_{iz}.$$

Consideremos un plano normal a la dirección y. La distancia de una esfera a dicho plano estará dada por un vector de magnitud δr_y, cuyas componentes en x y z son nulas. De manera análoga, la distancia de una esfera a un plano normal al eje Z tendrá componentes nulas en x e y. Operativamente, basta con forzar que δr_x y δr_z valgan cero para efectivamente simular un plano infinito paralelo al plano xz y que contiene al punto $\mathbf{r}_j$. Así pues, se puede definir directamente cualquier plano paralelo a los planos principales de la celda de simulación cancelando las compo-

nentes en las direcciones contenidas en el plano. Para definir un cilindro del mismo radio que su partícula asociada bastará con cancelar la coordenada correspondiente al eje del cilindro.

En el caso de un plano arbitrario, con una orientación que no coincida con ninguno de los ejes del marco *espacial* de referencia, se declara su orientación por medio de cuaterniones. A partir de estos valores, se calcula la matriz de rotación **A** que transforma del marco *espacial* a un marco *auxiliar* y fuerza al plano arbitrario a coincidir con el plano xz. Premultiplicando el vector $\mathbf{r}_{ij}$ que une los centros de ambas partículas—la partícula de referencia del plano y la partícula activa que queramos hacer interactuar con él—por la matriz de rotación **A**, obtenemos un nuevo vector $\mathbf{r}'_{ij}$ en el marco de referencia tal que el plano arbitrario coincide exactamente con el plano xz. Si ahora, olvidándonos de las otras dos componentes, calculamos la componente $\delta r'_y$ y la premultiplicamos por la matriz $\mathbf{A}^{-1}$, habremos obtenido un vector perpendicular al plano arbitrario que va de este último al centro de la partícula activa y que se encuentra expresado en el marco *espacial* de referencia.

Hasta ahora, el plano descrito es esencialmente infinito, pero con un pequeño defecto: si el plano atraviesa una frontera periódica, se rompe en segmentos que pueden causar severos conflictos (*Figura 11a* y *b*). Para evitar este problema, es necesario fijarnos en las dos componentes, $\delta r'_x$ y $\delta r'_z$, que originalmente ignoramos. El concepto de longitud es independiente del marco de referencia empleado. Como resulta más conveniente definir el tamaño del plano dentro del marco *auxiliar* de referencia, declaramos un par de valores x_{pln} y z_{pln} que representen la longitud total del plano en las direcciones x' y z'. En cuanto el valor absoluto de $\delta r'_x$ (o de $\delta r'_z$) exceda $\frac{1}{2}x_{pln}$ (o $\frac{1}{2}z_{pln}$, correspondientemente), dejamos de considerar a la partícula de referencia como un plano, para considerarla un cilindro paralelo al eje Z (o X, respectivamente). De esta manera hemos transformado a un plano infinito en un plano finito de área $x_{pln} \times z_{pln}$, que se encuentra delimitado por bordes redondeados (los cilindros). De esta manera, una vista en planta del plano obtenido se parecerá a la que se muestra (*Figura 11c*).

Una vez definidos los planos arbitrarios, se puede definir con facilidad cualquier eje de simetría. Para ello, consideremos un plano de simetría como un espejo. La forma tradicional en la que conocemos a un espejo es como un plano vertical[a] finamente pulido en el cual se reflejan las cosas colocadas frente a él. Los reflejos obtenidos tienen varias características. Primeramen-

[a] Para nuestros fines, consideraremos a éste como un plano normal al eje Y.

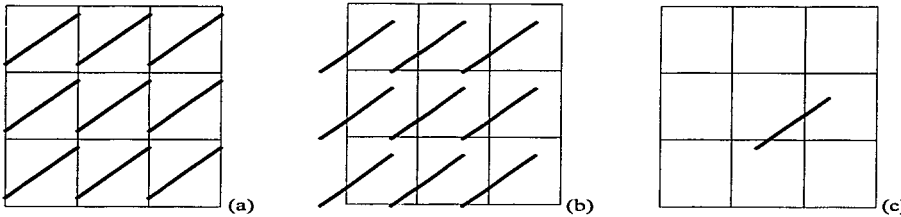


Figura 11. La figura (a) muestra el aspecto de un "plano infinito" dentro de fronteras periódicas, considerando que la partícula de referencia se encuentra al centro de la celda. Si la partícula se encuentra en otra posición, el efecto puede ser aún peor, como se muestra en la figura (b). La figura (c) muestra el resultado obtenido para un plano finito.

te, los desplazamientos en los ejes X y Z se reflejan sin modificación alguna, mientras que los desplazamientos en y se reflejan con signo opuesto. En segundo lugar, las rotaciones se reflejan de manera contraria a los desplazamientos, pues las rotaciones que cambian su signo en la imagen son las rotaciones en torno a x y z, mientras que las rotaciones en torno al eje Y conservan su mismo signo, como se muestra a continuación (*Figura 12*).

Parecería entonces que para simular un plano de simetría basta con colocar un plano y cambiar el signo de los desplazamientos, velocidades, rotaciones y velocidades angulares de cualquier partícula que interactúe con dicho plano, para así obtener su reflejo. Si asignamos las propiedades de la partícula reflejada a la partícula de referencia del plano, lograremos que una partícula que se aproxime al espejo interactúe con su propia imagen. Este enfoque es correcto para medios continuos, pero tiene un grave defecto para materiales granulares (sistemas discretos), como se muestra en la primera imagen de los resultados (*Figura 14*, pág. 41). La simetría obtenida es "dura" y provoca un relativo ordenamiento que se sostiene por una distancia de algunos diámetros. El problema radica no sólo en el ordenamiento, sino en que todas las partículas que están en contacto con el espejo se alinean perfectamente en la superficie de éste. En un sistema real, por muy simé-

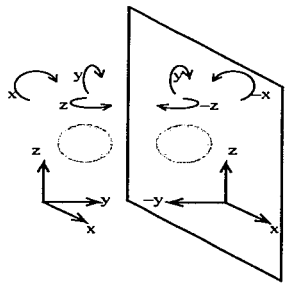


Figura 12. Grados de libertad de una partícula y su imagen reflejada en un espejo plano.

trico que sea, las partículas tienen la posibilidad de atravesar de un lado al otro del plano de simetría. Para acercarnos más a la realidad debemos permitir que las partículas penetren en mayor o menor grado la superficie del espejo.

Se propusieron dos soluciones para lograr una simetría más real: espejos *porosos* y espejos *brownianos*. La primera propuesta implica una cierta densidad de perforaciones distribuidas aleatoriamente sobre la superficie del plano, pero esto complica fuertemente la programación. La segunda implica un leve movimiento aleatorio del espejo en una dirección perpendicular a sí mismo y es directamente programable. Por simplicidad, se optó por la segunda propuesta.

Los llamados espejos *brownianos* toman su nombre del movimiento que se presenta en las partículas de un coloide. Operativamente, lo que se hace es asignar a cada par de vecinos cercanos un número aleatorio $0 \le n_b < 1$. Si al hacer el cálculo de fuerzas determinamos que una de las partículas en cuestión corresponde a un espejo, ajustamos la distancia entre ambas partículas con base en la fórmula

$$\delta r_y = r_{jy} - r_{iy} + (2n_b - 1)\rho_j$$

donde ρ_j es el radio de la partícula de referencia del espejo. De esta manera se obtiene un espejo roto a pedazos, cada uno de los cuales se mueve hacia adelante o atrás dependiendo de con cuál partícula interactúe, para formar un arreglo semejante al que se muestra (*Figura 13*). Como el ajuste de distancia se hace para cada par de vecinos cercanos en el instante en el que se detectan, se evita que las partículas choquen inesperadamente con un espejo que aparece de pronto.

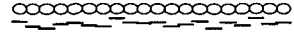


Figura 13. Se muestra un espejo *browniano* al cual se aproximan 20 partículas perfectamente alineadas. Nótese que el espejo asume una posición diferente para cada partícula, pero su desplazamiento en ningún caso excede un diámetro de partícula.

El desplazamiento del espejo *browniano* está dictado por el valor de n_b, que a su vez proviene de un generador de números aleatorios uniformemente distribuidos. Esto hace que el espejo se encuentre en la misma posición promedio para todas las partículas, por lo que su cambio real de posición para cada par de vecinos no afecta a la generalidad del sistema. Los resultados obtenidos con este tipo de simetría se ajustan muy bien a la realidad (*Figura 15*, pág. 42).

La compilación del código completo, incluyendo las modificaciones, se realizó en una estación de trabajo Hewlett Packard, empleando el compilador `f77` (Fortran77) de la versión 10.01 de HP-UX. Para garantizar el mejor desempeño posible, se compiló utilizando el máximo grado de optimización posible. Con esto, el tiempo de compilación aumenta unas ocho veces con respecto al tiempo necesario para compilar sin optimización de código, *i.e.* el tiempo sube de poco más de un minuto a cerca de diez. Esta inversión de recursos en la compilación paga con creces la demora, pues la velocidad de ejecución del programa (comparando la compilación optimizada con la compilación sin optimizar) aumenta en un 78%.[a] El código fuente tiene una extensión de 175 KB, mientras que el archivo ejecutable obtenido mide aproximadamente 150 KB.

Referencias

[1] Walton y Braun, 1986.

[2] Allen y Tildesley, 1987.

[3] Evans y Murad, 1977.

[4] Goldstein, 1950.

[5] Allen y Tildesley, 1987.

[6] Timoshenko y Goodier, 1970.

[7] Mindlin, 1949.

[a] Valor promedio obtenido de 12 corridas de prueba.

Resultados Obtenidos y Discusión

Para poder evaluar las modificaciones al código original conforme se iban realizando, se hicieron varias corridas, cuyos archivos de entrada se muestran en el Apéndice B. En todos los casos se emplearon partículas con una densidad igual a la del maíz, de 1200 kg/m^3, lo que equivale a 720 kg/m^3 de densidad a granel (con una fracción de empaquetamiento $\eta = 0.6$).[1] La configuración inicial dentro de la celda de simulación es aleatoria y uniformemente distribuida. Los cálculos y compilación se realizaron en una estación de trabajo Hewlett Packard 9000/712 a 80 MHz, con 48 MB de memoria central, corriendo HP-UX 10.01.

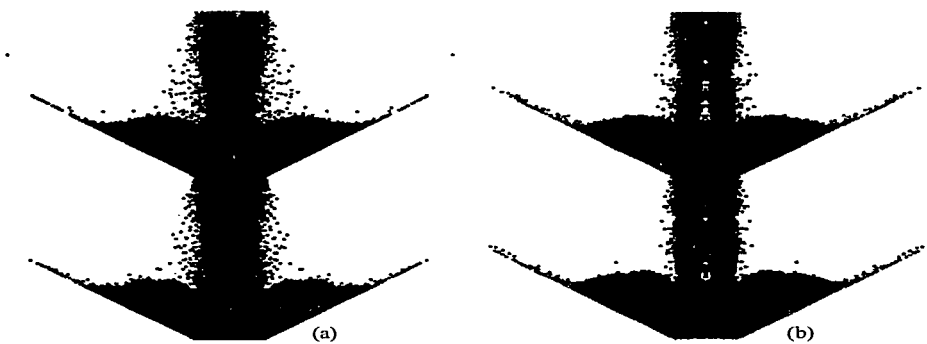


Figura 14. La figura (a) muestra la primera simulación obtenida con un plano de simetría. La simetría es "dura" y las partículas adyacentes al espejo se alinean perfectamente, como si estuvieran apoyadas en una pared rígida. Si en vez de un espejo hubiéramos empleado una pared rígida, las partículas rebotarían en ella, formando oquedades como se muestra en la figura (b). Cada imagen consta de 14 000 partículas (7000×2).

Una vez definido el concepto de espejos para sistemas con simetría, se hizo una corrida de prueba (Corrida 1, Apéndice B) en la que se colocaron dentro de la celda dos planos inclinados a 35 grados respecto a la horizontal, como una primera aproximación a un silo hexagonal.[2] La celda de simulación tiene un ancho de 1.1625 m, un alto de 1.55 m y un espesor (en la dirección perpendicular al papel) de 2.325 cm. Las fronteras laterales son una pared rígida y un espejo, mientras que en las direcciones vertical y frontal se cuenta con fronteras periódicas, *i.e.* las partículas que salen por la parte inferior (frontal o posterior) entran por la parte superior (posterior o frontal, respectivamente) de la celda. En esta corrida las partículas tiene un radio de 4.9 mm, que es el radio equivalente de un grano de maíz.[3] Se observa una simetría extremadamente falsa y antinatural, en la que las partículas adyacentes al eje de simetría se encuentran perfectamente alineadas con éste (*Figura 14a*). El resultado obtenido difiere, sin embargo, del que se extraería de emplear una pared rígida en lugar del espejo, como se muestra en la imagen (*Figura 14b*).

Indudablemente esta manera no era la más adecuada para simular el sistema, por lo que, revisando el código, se introdujo el concepto de espejos *brownianos*, que ya se describió previamente. Empleando el mismo archivo de entrada, pero con el nuevo código, *i.e.* espejos *brownianos*, se repitió la corrida, obteniendo el resultado que se muestra (*Figura 15*). Aquí la simetría se apega muy bien a la realidad.

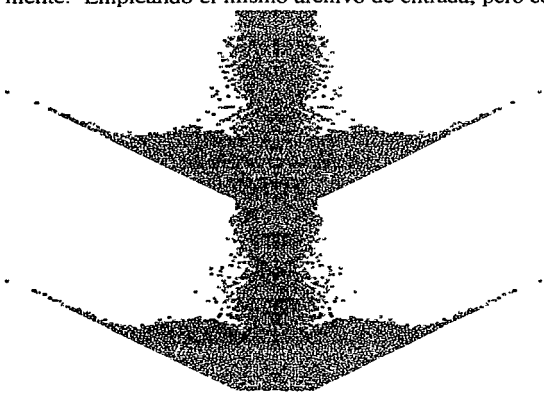


Figura 15. La misma simulación de la figura anterior, pero evaluándola con un plano de simetría *browniano*. El resultado es mucho más convincente.

Para probar la independencia del silo hexagonal al factor de escala,[4] es decir, ante partículas de diferentes tamaños y geometrías, se hicieron varias corridas más (Corrida 2, Apéndice B), cambiando el radio pero manteniendo constante la densidad y demás parámetros físicos. De esta manera se determinó que, para el silo de tres toneladas,[5] cuyas dimensiones ya se mencionaron para la corrida anterior—se duplicó, sin embargo, el

espesor, para hacerlo igual a 4.65 cm—, el radio máximo para las partículas es de 1 cm. Para cualquier radio menor al mencionado, el comportamiento del flujo es siempre el mismo, mientras que para un radio mayor se observa que el desfogue se bloquea por un efecto de "cuello de botella". A continuación se muestra una imagen con partículas de 1 cm de radio (*Figura 16*). Nótese que en la parte inferior de la imagen hay unas cuantas partículas, aparentemente flotando. Debido a que las fronteras superior e inferior son periódicas, estas partículas corresponden a la parte superior. Con esta consideración en mente, observamos que las mitades superior e inferior de la celda de simulación tienen configuraciones totalmente correspondientes entre sí.

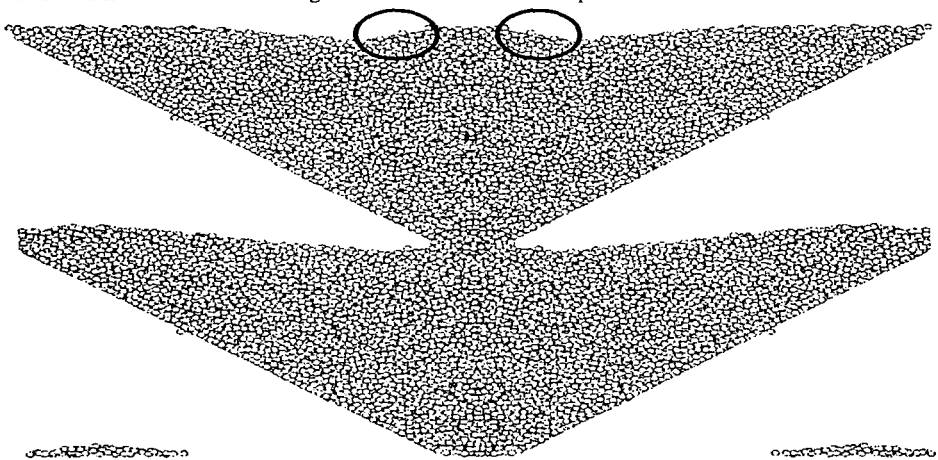
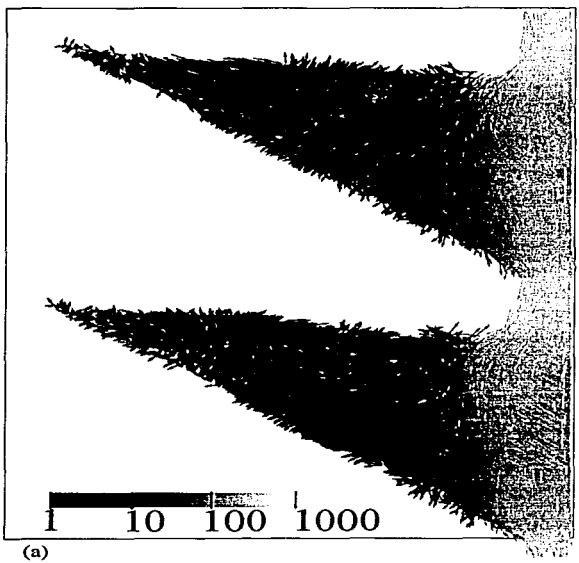


Figura 16. Simulación con 14 000 partículas (7000×2) de 1 cm de radio. Los pequeños montículos en la parte inferior corresponden, debido a las fronteras periódicas, a la parte más alta de la mitad superior. Los ángulos dinámicos de reposo (encerrados con un círculo) corresponden a las esferas con coeficiente de fricción $\mu = 0.5$.[6]

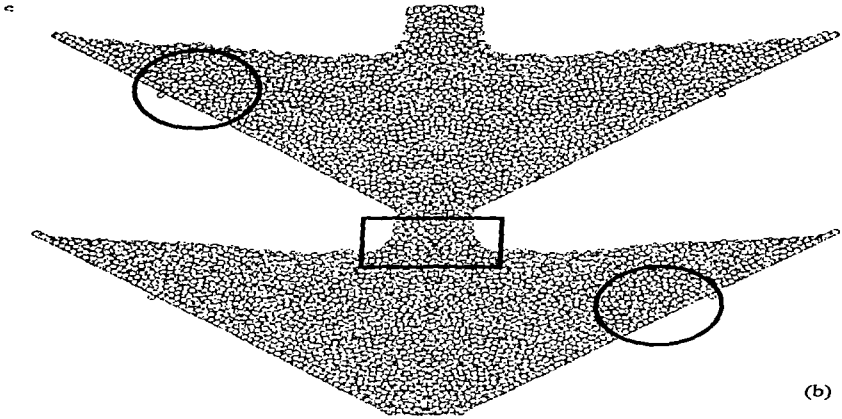
A estas alturas ya hemos determinado el diámetro máximo de las partículas para la simulación. Este valor es crucial pues nos permite obtener resultados confiables con el menor número posible de partículas, lo cual es muy importante para minimizar el tiempo de cómputo requerido. Sin embargo, estamos descuidando un detalle igualmente importante. Aún con fronteras periódi-

cas, es recomendable que el espesor de la celda de simulación sea cuando menos de cinco diámetros de partícula, para que no se favorezca un acomodo cristalino. Tomando esto en cuenta, se realizó una corrida más (Corrida 3, Apéndice B), aumentando el espesor de la celda a 11.625 cm y el número de partículas a 12 000. Las figuras siguientes (*Figura 17a–c*) muestran las posiciones y distribución de velocidades de las partículas dentro del silo en estado permanente. El estado permanente sólo se puede obtener en una simulación, pues implica tener una masa constante que es posible exclusivamente si logramos que la masa que sale por un lado entre por el otro, *i.e.* fronteras periódicas. El mapa de velocidades (al igual que un análisis detallado de las posiciones en los dos instantes representados) muestra que el flujo principal se presenta en un núcleo central que desciende sin casi ninguna contribución de los lados.

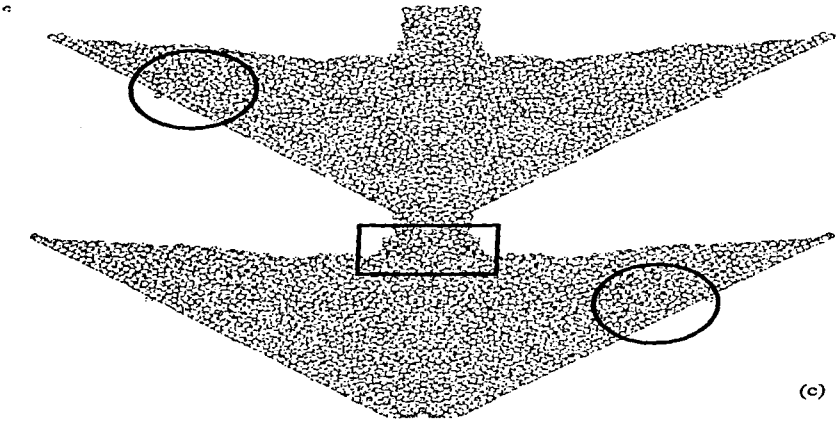


(a)

Figura 17. La figura (a) muestra el campo de velocidades de las partículas dentro de la celda de simulación en estado permanente. La celda se muestra con la línea continua. Los tonos más claros corresponden a velocidades más altas. La velocidad del núcleo central (extremo derecho de la celda) es entre dos y tres órdenes de magnitud mayor que la de las partículas sobre la rampa (a la izquierda). Los tonos de gris representan el logaritmo de la rapidez de las partículas en cada punto. Las figuras (b) y (c), a la derecha, muestran las posiciones de 24 000 (12 000×2) partículas de 1 cm de radio, a 1 y 4 segundos de alcanzado el estado permanente. Un análisis detallado de las posiciones permite ver que la configuración en la rampa prácticamente no cambia (como es el caso de las zonas encerradas por círculos), mientras que el núcleo central modifica substancialmente su configuración, al grado de que no se puede decir cuales partículas son la misma en ambas imágenes (zona encerrada por rectángulos).



(b)



(c)

En este punto ya se han determinado todas las condiciones geométricas del flujo, y se ha establecido una clara analogía entre el comportamiento de la simulación numérica y el comportamiento real del flujo dentro del silo hexagonal durante el proceso de vaciado.[7] Con la seguridad de que tanto el algoritmo como la descripción geométrica empleadas son correctos, sólo resta hacer una simulación del silo completo, para la totalidad de su llenado y de su vaciado. A continuación se presenta una imagen (*Figura 18*) de la configuración de las partículas dentro del silo totalmente lleno (Corrida 4, Apéndice B). En este caso, la celda mide 2 m de alto, pues se trata del silo completo y no sólo de la celda inferior. El espesor de la rebanada empleada es de 12.767 cm y el tiempo de cómputo empleado para calcular la configuración mostrada es de cerca

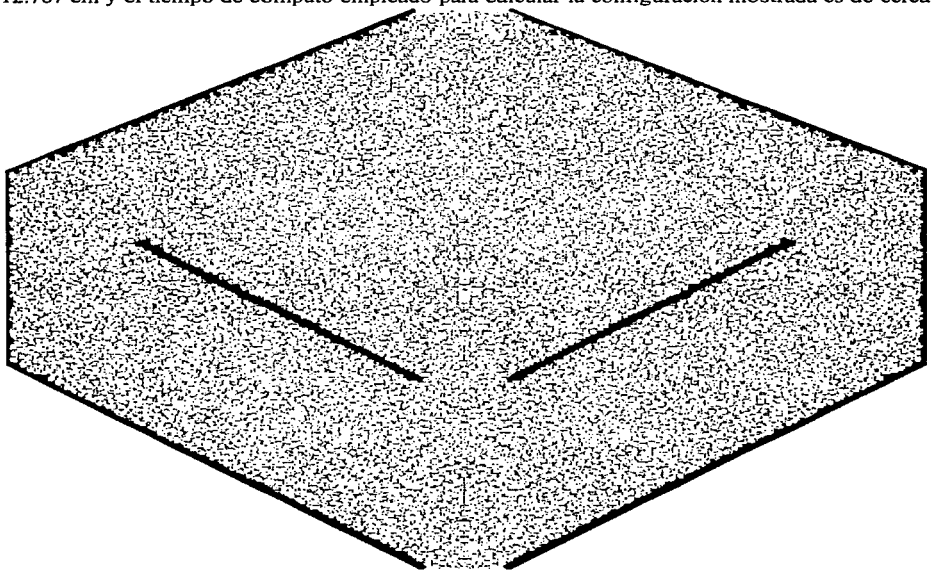


Figura 18. Imagen simulada del silo hexagonal totalmente lleno. Se muestran 50 000 (25 000×2) partículas del máximo diámetro permisible. Empleando partículas con un diámetro equivalente al del maíz, se hubiera requerido de 430 000 (215 000×2), lo que significaría un tiempo de cómputo 70 veces mayor.

de tres horas. Se muestra la celda sin flujo pues el tiempo de cómputo necesario para calcular el vaciado del silo en una estación de trabajo serial se vuelve prohibitivo. Para el caso de la HP 712 empleada, se requeriría aproximadamente de 140 semanas[a] de cómputo ininterrumpido para calcular la totalidad del vaciado. Con un poco de paralelización en el código y el empleo de una supercomputadora (como la CRAY-YMP 4/464 o la Origin-2000) se podría reducir este tiempo entre 100 y 250 veces[b], lo que significa que el cálculo del silo completo tomaría unos cuantos días. Se deduce de aquí que, a menos que se empleen recursos de supercómputo o se simule por pedazos, resulta imposible simular el proceso completo de vaciado del silo. Lo mismo se puede decir respecto al llenado, pues se requiere dar seguimiento a todo el proceso para poder definir con exactitud la configuración que se obtiene.[8]

Referencias

[1] Chávez Montes, 1997; Galicia Ávila, 1997.

[2] Hernández, Morano y Sosa, 1995.

[3] Brooker *et. al.*, 1992.

[4] Joseph *et. al.*, 1996.

[5] Hernández, Morano y Sosa, 1995.

[6] Walton y Braun, 1993.

[7] Hernández, Morano y Sosa, 1995.

[8] Hernández, Morano y Sosa, 1995.

[a] Tiempo estimado con base en una simulación de una semana, que equivale a algo menos de 0.2 s de tiempo real.

[b] Valores estimados con base en el desempeño en punto flotante de las máquinas y en el grado de paralelización del algoritmo.

Conclusiones

Se comprobó que el algoritmo de dinámica molecular propuesto por el Dr. Otis Walton es capaz de simular, con ciertas modificaciones importantes, el flujo dentro del silo hexagonal inventado por el Dr. Baltasar Mena.

Se determinó que los planos arbitrarios infinitos producen conflictos con las fronteras periódicas de la simulación numérica; los planos finitos, en cambio, son fáciles de controlar y no producen conflictos.

Para simular planos finitos adecuados, es necesario que sus bordes tengan geometría cilíndrica, evitando así que las partículas que se aproximan hacia las orillas se encuentren de pronto con un plano, o al contrario, en caso de que el desplazamiento sea en sentido opuesto, que el plano desaparezca.

La simetría de un sistema granular se puede simular asignando propiedades de espejo a un plano arbitrario.

Los espejos para simulaciones de sistemas granulares simétricos no pueden ser planos, pues en ese caso se obtienen resultados erróneos. Se requiere de espejos porosos o de espejos *brownianos*, siendo estos últimos los más fáciles de programar.

Vale la pena invertir un tiempo largo en la compilación, con el objeto de obtener un archivo ejecutable más eficiente.

Para simular sistemas granulares complejos (mayores que 10 000 partículas), las estaciones de trabajo convencionales resultan insuficientes. Se requiere de cómputo paralelo o de supercómputo para simular exitosamente sistemas tales como el silo completo.

¿Qué sigue?

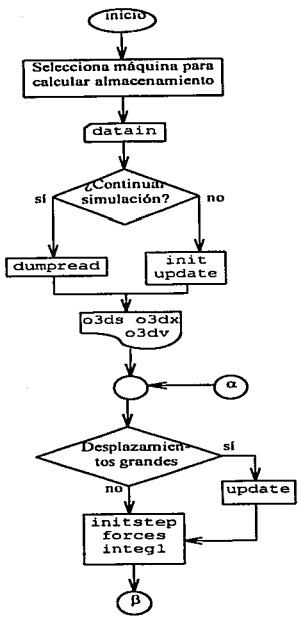
Se sugiere llevar a cabo una serie de simulaciones del silo completo, empleando recursos de supercómputo, para comparar los resultados con las filmaciones del vaciado y llenado con las que actualmente se cuenta.

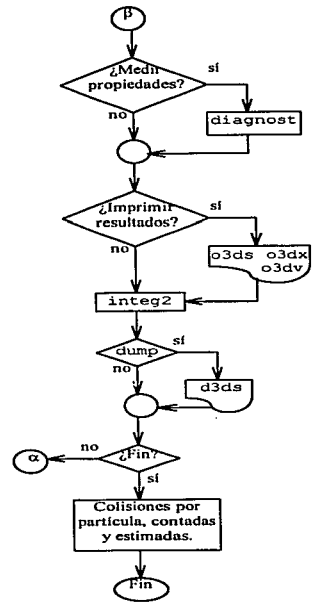
Dado que la geometría del silo se ha simulado de manera exitosa, es de esperarse que el programa arroje resultados correctos en cuanto a presión, esfuerzos y empaquetamiento. Se recomienda hacer varias corridas para determinar estas variables y comparar los resultados con las mediciones realizadas en el modelo de laboratorio y en el silo piloto.

Se plantea también como labor futura emplear cúmulos[a] de esferas para simular con mayor precisión la geometría de granos alimenticios como el maíz.

[a] En inglés, *clusters*.

Código Fuente





```
c
c-------------------------------------------------------------------------
c
c     s3dscmn
c
c----machine-dependent parameter
c     ivers = 1 for cray computer
c     ivers = 2 for sun workstation
c-----  note cray (cft) fortran does not recognize integer*4 but instead
c       automatically uses 64 bits for all integers
      parameter (ivers=2)
c                               nwd is 9 for cray or 8 for sun
      parameter (nwd=10-ivers)
c
c-----parameters
      parameter (mp=30100,nvelmx=10,myzone=21,mnave=10,mgroup=6)
c
      implicit real*8 (a-h,o-z)
      real*8 mass,massi,masst,massti
      character*1 nchar
      character*8 labcell,labz,labg,title,oldttl,lastchr,version
      integer*4 ndx,next
c
c     lcm linklist
c-----      variables `ndx' and `next' must be adjacent to each other
c           on machines that recognize 32-bit integers
c
      common/linklist/ ndx(1),next(1),a(1),a0(1),fn(1),tfx(1),tfy(1)
     1    ,tfz(1),tm(1 + nwd*mp*mnave)
c
c-----character variables
c
      common/chars/version,title(10),oldttl(10)
     >  ,labcell,labz(myzone),labg(mgroup),nchar(80),lastchr
c
c-----note: start storing scm common in dumpfile at variable firstsc
      common firstsc(1)
c
      common tmax,dt,dtout,dtoutx,dtoutv,tzero,edot,epsv
      common vavez,vseed,draddt
      common skn1,elast,slope,skn1b,elastb,slopeb,dash1,dash2
      common ratk,fmu,power,rmassz,fmub,drag,dash1b,dash2b,finis
      common gravx,gravy,gravz
      common vxzero,vyzero,vzzero,wxzero,wyzero,wzzero,wzcyl
      common vxby0,vxbyl,vyby0,vybyl,vzby0,vzbyl
      common vxbz0,vxbz1,vybz0,vybz1,vzbz0,vzbz1
      common vcell,xcell,ycell,zcell,xyrat,zyrat
      common vcell1,xcell1,ycell1,zcell1,xcell1h,ycell1h,zcell1h
      common ystart,ystop,dyzone,dyzon1,vyzone,vyzoni
c
      common pi,ccpack
      common half,third,fourth,fifth,sixth,two5ths,threehf
      common one,two,three,four,ten,trifle,huge
c
      common t,tout,toutx,toutv,sigma,edoti
      common rmax,rmin,rave,skn2,skn2b
      common delvx,delvy,delvz,delvxi,delvyi,delvzi
      common delwx,delwy,delwz,delwxi,delwyi,delwzi
      common colfrq,totcol,dum,yfact,dmass,xmax
      common vsqnz,tshift,rmassx,rmassy,packf,packbf
      common vfree,vbound,vfixed,vzcyl,vtotal
      common packy0,packy1,packz0,packz1,packfx,packcz
c
      common save,savei,savet,saveti
      common massi,massti,vavei,vaveti
      common dvsqn,dxmom,dspnx,dspny,dspnz,dekin,depot,derot
      common dgekin(mgroup),dgerot(mgroup)
      common dgflox(mgroup),dgfloy(mgroup),dgfloz(mgroup)
c
      common vsqn,xmom,spnx,spny,spnz
      common pack,ekin,epot,erot
      common comp,visc,mass,vave
      common vxave,vyave,vzave,wzave
      common gekin(mgroup),gerot(mgroup)
      common gflox(mgroup),gfloy(mgroup),gfloz(mgroup)
      common gvxave(mgroup),gvyave(mgroup),gvzave(mgroup)
c
      common vsqnt,xmomt,spnxt,spnyt,spnzt
      common packt,ekint,epott,erott
      common compt,visct,masst,vavet
      common gekint(mgroup),gerott(mgroup),gwzave(mgroup)
      common gfloxt(mgroup),gfloyt(mgroup),gflort(mgroup)
c
      common dymass(myzone),dyvsqn(myzone),dyxmom(myzone)
      common dyspnx(myzone),dyspny(myzone),dyspnz(myzone)
      common dyedot(myzone),dypack(myzone),dyekin(myzone),dyepot(myzone)
      common dyerot(myzone)
c
      common ymass(myzone),yvsqn(myzone),yxmom(myzone)
      common yspnx(myzone),yspny(myzone),yspnz(myzone)
      common yedot(myzone),ypack(myzone),yekin(myzone),yepot(myzone)
      common yerot(myzone),ycomp(myzone),yvisc(myzone),yvave(myzone)
c
      common ymasst(myzone),yvsqnt(myzone),yxmomt(myzone)
      common yspnxt(myzone),yspnyt(myzone),yspnzt(myzone)
      common yedott(myzone),ypackt(myzone),yekint(myzone),yepott(myzone)
      common yerott(myzone),ycompt(myzone),yvisct(myzone),yvavet(myzone)
c
      common yvx(myzone),yvz(myzone)
      common ymassi(myzone),ymassti(myzone)
      common yvavei(myzone),yvaveti(myzone)
c
      common dpnnk(9),dpnnp(9),pnnk(9),pnnp(9),pnnkt(9),pnnpt(9),rnfn(9)
      common dypnnk(myzone,9),dypnnp(myzone,9)
      common ypnnk(myzone,9),  ypnnp(myzone,9)
      common ypnnkt(myzone,9),ypnnpt(myzone,9)
c
      common dpxwal,dpywal,dpzwal,pxwall,pywall,pzwall
      common pxwalt,pywalt,pzwalt
c
      common radius(mgroup),pmass(mgroup),planex(mgroup),planez(mgroup)
      common radmax(mp),radz(mp),rad(mp),xplnh(mp),zplnh(mp)
      common brown(mp),rmass(mp),rmnt(mp),vol(mp),time(mp)
      common  x(mp),  y(mp),  z(mp)
      common dx(mp),dy(mp),dz(mp)
      common xp(mp),yp(mp),zp(mp)
c
      common q1old(mp),q2old(mp),q3old(mp),q4old(mp)
      common q1new(mp),q2new(mp),q3new(mp),q4new(mp)
c
      common vx(mp),vy(mp),vz(mp),wx(mp),wy(mp),wz(mp)
      common fx(mp),fy(mp),fz(mp),ftx(mp),fty(mp),ftz(mp)
```

```
      common dvx(mp),dvy(mp),dvz(mp),dwx(mp),dwy(mp),dwz(mp)
      common vhx(mp),vhy(mp),vhz(mp),whx(mp),why(mp),whz(mp)
      common tvx(mp),tvy(mp),tvz(mp)
c
      common search,tupdtl,delrup,drmax
c
      common vpx(mp),vnx(mp)
      common rpos(mp),rneg(mp),ypshft(mp),ynshft(mp)
c
      common binvx(nvelmx),binvy(nvelmx),binvz(nvelmx)
      common binvxt(nvelmx),binvyt(nvelmx),binvzt(nvelmx)
c
      common binwx(nvelmx),binwy(nvelmx),binwz(nvelmx)
      common binwxt(nvelmx),binwyt(nvelmx),binwzt(nvelmx)
c
      common gbinvx(nvelmx,mgroup),gbinvy(nvelmx,mgroup)
      common gbinvz(nvelmx,mgroup),gbinvxt(nvelmx,mgroup)
      common gbinvyt(nvelmx,mgroup),gbinvzt(nvelmx,mgroup)
c
      common epsrep,sigrep,rijsqn,epotz
      common zeta,dtdump,tdump,tstart
c
c-----integers
      common nrun,info(2)
      common mtl,maxwd,nebor(mp)
      common npos(mp),nneg(mp),number(mgroup)
      common np,nby0,nby1,nbz0,nbz1,nfix,nfxpl,nzcyl
      common nxby0,nzby0,nxby1,nzby1,nxbz0,nybz0,nxbz1,nybz1
      common ncut,noutx,nouty,nczero,ncmax,ntcol,nvel,nyzone
      common ireal,imirr
      common iterun,izeta,icoord,iquat,itty,ihertz,istart,ialtk,istop
      common izero,itervs,nvelhalf,nstep,irad,iextra,imin,imax,itot
      common ind1,ind2,nminus,nplus,imove,nshift,mshift,ngroup,icell
      common ixyz,ndx2,ind1y0,ind2y0,ind1y1,ind2y1
      common ind1z0,ind2z0,ind1z1,ind2z1
      common ind1fx,ind2fx,ind1fxp,ind2fxp,ind1cz,ind2cz
      common ndump,lld,len,i2or1,i1or0,i8or1
c
c-----note: stop storing common in dumpfile at variable lastsc
      common idummy, lastsc, idummy2
      common lenfp,lenchr,ilast11, llused, illused
c
c
      dimension dypxxk(nyzone),dypyxk(nyzone),dypzxk(nyzone)
     1         ,dypxyk(nyzone),dypyyk(nyzone),dypzyk(nyzone)
     2         ,dypxzk(nyzone),dypyzk(nyzone),dypzzk(nyzone)
c
      dimension dypxxp(nyzone),dypyxp(nyzone),dypzxp(nyzone)
     1         ,dypxyp(nyzone),dypyyp(nyzone),dypzyp(nyzone)
     2         ,dypxzp(nyzone),dypyzp(nyzone),dypzzp(nyzone)
c
      dimension ypxxk(nyzone),ypyxk(nyzone),ypzxk(nyzone)
     1         ,ypxyk(nyzone),ypyyk(nyzone),ypzyk(nyzone)
     2         ,ypxzk(nyzone),ypyzk(nyzone),ypzzk(nyzone)
c
      dimension ypxxp(nyzone),ypyxp(nyzone),ypzxp(nyzone)
     1         ,ypxyp(nyzone),ypyyp(nyzone),ypzyp(nyzone)
     2         ,ypxzp(nyzone),ypyzp(nyzone),ypzzp(nyzone)
c
      dimension ypxxkt(nyzone),ypyxkt(nyzone),ypzxkt(nyzone)
     1         ,ypxykt(nyzone),ypyykt(nyzone),ypzykt(nyzone)
     2         ,ypxzkt(nyzone),ypyzkt(nyzone),ypzzkt(nyzone)
c
      dimension ypxxpt(nyzone),ypyxpt(nyzone),ypzxpt(nyzone)
     1         ,ypxypt(nyzone),ypyypt(nyzone),ypzypt(nyzone)
     2         ,ypxzpt(nyzone),ypyzpt(nyzone),ypzzpt(nyzone)
c
c
      equivalence (rxfx,rnfn(1)),(ryfx,rnfn(2)),(rzfx,rnfn(3))
     1           ,(rxfy,rnfn(4)),(ryfy,rnfn(5)),(rzfy,rnfn(6))
     2           ,(rxfz,rnfn(7)),(ryfz,rnfn(8)),(rzfz,rnfn(9))
c
      equivalence (dpxxk,dpnnk(1)),(dpyxk,dpnnk(2)),(dpzxk,dpnnk(3))
     1           ,(dpxyk,dpnnk(4)),(dpyyk,dpnnk(5)),(dpzyk,dpnnk(6))
     2           ,(dpxzk,dpnnk(7)),(dpyzk,dpnnk(8)),(dpzzk,dpnnk(9))
c
      equivalence (dpxxp,dpnnp(1)),(dpyxp,dpnnp(2)),(dpzxp,dpnnp(3))
     1           ,(dpxyp,dpnnp(4)),(dpyyp,dpnnp(5)),(dpzyp,dpnnp(6))
     2           ,(dpxzp,dpnnp(7)),(dpyzp,dpnnp(8)),(dpzzp,dpnnp(9))
c
      equivalence (pxxk,pnnk(1)),(pyxk,pnnk(2)),(pzxk,pnnk(3))
     1           ,(pxyk,pnnk(4)),(pyyk,pnnk(5)),(pzyk,pnnk(6))
     2           ,(pxzk,pnnk(7)),(pyzk,pnnk(8)),(pzzk,pnnk(9))
c
      equivalence (pxxp,pnnp(1)),(pyxp,pnnp(2)),(pzxp,pnnp(3))
     1           ,(pxyp,pnnp(4)),(pyyp,pnnp(5)),(pzyp,pnnp(6))
     2           ,(pxzp,pnnp(7)),(pyzp,pnnp(8)),(pzzp,pnnp(9))
c
      equivalence (pxxkt,pnnkt(1)),(pyxkt,pnnkt(2)),(pzxkt,pnnkt(3))
     1           ,(pxykt,pnnkt(4)),(pyykt,pnnkt(5)),(pzykt,pnnkt(6))
     2           ,(pxzkt,pnnkt(7)),(pyzkt,pnnkt(8)),(pzzkt,pnnkt(9))
c
      equivalence (pxxpt,pnnpt(1)),(pyxpt,pnnpt(2)),(pzxpt,pnnpt(3))
     1           ,(pxypt,pnnpt(4)),(pyypt,pnnpt(5)),(pzypt,pnnpt(6))
     2           ,(pxzpt,pnnpt(7)),(pyzpt,pnnpt(8)),(pzzpt,pnnpt(9))
c
      equivalence (dypxxk,dypnnk(1,1)),(dypyxk,dypnnk(1,2))
     1           ,(dypzxk,dypnnk(1,3)),(dypxyk,dypnnk(1,4))
     1           ,(dypyyk,dypnnk(1,5)),(dypzyk,dypnnk(1,6))
     1           ,(dypxzk,dypnnk(1,7)),(dypyzk,dypnnk(1,8))
     4           ,(dypzzk,dypnnk(1,9))
c
      equivalence (dypxxp,dypnnp(1,1)),(dypyxp,dypnnp(1,2))
     1           ,(dypzxp,dypnnp(1,3)),(dypxyp,dypnnp(1,4))
     1           ,(dypyyp,dypnnp(1,5)),(dypzyp,dypnnp(1,6))
     3           ,(dypxzp,dypnnp(1,7)),(dypyzp,dypnnp(1,8))
     4           ,(dypzzp,dypnnp(1,9))
c
      equivalence (ypxxk,ypnnk(1,1)),(ypyxk,ypnnk(1,2))
     1           ,(ypzxk,ypnnk(1,3)),(ypxyk,ypnnk(1,4))
     2           ,(ypyyk,ypnnk(1,5)),(ypzyk,ypnnk(1,6))
     3           ,(ypxzk,ypnnk(1,7)),(ypyzk,ypnnk(1,8))
     4           ,(ypzzk,ypnnk(1,9))
c
      equivalence (ypxxp,ypnnp(1,1)),(ypyxp,ypnnp(1,2))
     1           ,(ypzxp,ypnnp(1,3)),(ypxyp,ypnnp(1,4))
     2           ,(ypyyp,ypnnp(1,5)),(ypzyp,ypnnp(1,6))
     1           ,(ypxzp,ypnnp(1,7)),(ypyzp,ypnnp(1,8))
     4           ,(ypzzp,ypnnp(1,9))
c
      equivalence (ypxxkt,ypnnkt(1,1)),(ypyxkt,ypnnkt(1,2))
     1           ,(ypzxkt,ypnnkt(1,3)),(ypxykt,ypnnkt(1,4))
```

```
     2           ,(ypyykt,ypnnkt(1,5)),(ypzykt,ypnnkt(1,6))
     3           ,(ypxzkt,ypnnkt(1,7)),(ypyzkt,ypnnkt(1,8))
     4           ,(ypzzkt,ypnnkt(1,9))
c
      equivalence (ypxxpt,ypnnpt(1,1)),(ypyxpt,ypnnpt(1,2))
     1           ,(ypzxpt,ypnnpt(1,3)),(ypxypt,ypnnpt(1,4))
     2           ,(ypyypt,ypnnpt(1,5)),(ypzypt,ypnnpt(1,6))
     3           ,(ypxzpt,ypnnpt(1,7)),(ypyzpt,ypnnpt(1,8))
     4           ,(ypzzpt,ypnnpt(1,9))
```

```
      program s3ds
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c                  3dshear Version 2.05b, Feb 1, 1995                  c
c                 Otis R. Walton and Robert L. Braun                   c
c               Lawrence Livermore National Laboratory                 c
c                  P. O. Box 808  Mailstop L-207                       c
c                       Livermore, CA 94550                            c
c                                                                      c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c  Copyright to this work subsists at the Lawrence Livermore National  c
c  Laboratory under contract W-7405-ENG-48 between the Regents of the  c
c  University of California and the U.S. Department of Energy.  Upon   c
c  completion of field evaluation of this work, the University of      c
c  California will seek permission to claim the copyright as:         c
c        COPYRIGHT 1994, REGENTS OF THE UNIVERSITY OF CALIFORNIA       c
c  The recipient is not to further transfer the work or any derivative c
c  work without permission from the LLNL Technology Transfer Office,  c
c  use of this work or derivative works is limited to development and  c
c  evaluation, and the work should be considered an unpublished work.  c
c                                                                      c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c-----log of code changes:

c-----version s3ds5192, july 11, 1985 (from version .s:s2ds:s2ds5142uu)
c
c-----version s3ds5210, july 29, 1985 (from version .s:s3ds:s3ds5192uu)
c     use quaternions for representing orientation of particles
c
c-----version s3ds5213, aug. 1, 1985 (from version .s:s3ds:s3ds5210uu)
c     use more compact solution for quaternion equations
c
c-----version s3ds5221, aug. 9, 1985 (from version .s:s3ds:s3ds5213uu)
c     allout input of parameters epsrep and sigrep for 12th power repulsion
c
c-----version s3ds5228, aug. 16, 1985 (from version .s:s3ds:s3ds5221uu)
c     increase parameters ms=256 and mnave=12
c     new option izeta=2 for rescaling velocities to constant temperature
c     increase cutoff to sigrep=2 for 12th power law
c
c-----version s3ds5251, sept. 8, 1985 (from version .s:s3ds:s3ds5228uu)
c     replace initial cubic-close-packing algorithms with random packing
c     initialized particle radii to zero; expand radii at time zero
c     until particles are in contact; then continue to expand radii
c     each time step at rate 'draddt' until rad(i) = radz(i).
c     after radii expansion is finished, reset time to zero and
c     reinitialize other parameters.
c
c-----version s3ds5302, oct. 29, 1985 (from version .s:s3ds:s3ds5251uu)
c     option for run number on execution line.
c     name all input and output files with 3-digit run number.
c     allow maximum size for output files.
c
c-----version s3ds5311, nov. 8, 1985 (from version .s:s3ds:s3ds5302uu)
c     add tangential forces, store vector force (tfx,tfy,tfz) in link
```

```
c     list storage, change nwd to 9 words in each entry in list.
c     in subroutine findrad, increase the number of extra time steps
c     after radii expansion to 100.
c     change collision-frequency calculation in datasave and main.
c
c
c-----version s3ds5329, nov. 25, 1985 (from version .s:s3ds:s3ds5311uu)
c     sign changes in calculating torque on each particle
c
c-----version s3ds5347, dec. 13, 1985 (from version .s:s3ds:s3ds5329uu)
c     correct calculation of spin angular momenta
c
c-----version s3ds6073, mar. 14, 1986 (from version .s:s3ds:s3ds5347uu)
c     recalculate skn2 in subroutine forces, delete from link-list,
c     change nwd to 8, add velocity-dependent damping to hertz force model.

c-----version s3ds6080, mar. 21, 1986 (from version .s:s3ds:s3ds6073uu)
c     save old normal force in link-list (nwd=9);
c     add second velocity-dependent damping to hertz model if slope.lt.0.;
c     change subroutine findrad so that skn2 = skn1.
c
c-----version s3ds6101 apr. 11, 1986 (from version .s:s3ds:s3ds6080uu)
c     change calculation of rotational kinetic energy to use z-rotational
c     velocity relative to shear rate for whole cell (i.e., wz - 0.5*edot).
c     change calculation of rotational kinetic energy in y zone to
c     use z-rotational velocity relative to mean wz in that zone.
c     change calculation of compressibility factor to use
c     (2/3) kinetic energy.
c     change calculation of mean particle radius (rave) and diameter
c     (sigma) to use arithmetic, volume-weighted average.
c
c-----version s3ds6126, may 6, 1986 (from version .s:s3ds:s3ds6101uu)
c     revise subroutine bound to place boundary particles on xz plane(s) in
c     expanded, hexagonal-close-packed array
c
c-----version s3ds6135, may 15, 1986 (from version .s:s3ds:s3ds6126uu)
c     incorporate alternative way of inputing particle size and mass data,
c     specifying the number of particles having a given radius and rmass
c
c-----version s3ds6143, may 23, 1986 (from version .s:s3ds:s3ds6135uu)
c     apply shear correction to dx in subroutine integ2
c
c-----version s3ds6147, may 27, 1986 (from version .s:s3ds:s3ds6143uu)
c     change algorithm for initial velocities so that particles have
c     zero initial net momentum even when unequal-mass particles are used
c
c
c-----version s3ds6150, may 30, 1986 (from version .s:s3ds:s3ds6147uu)
c     diagnostics for kinetic and rotational energy for particle size groups
c
c-----version s3ds6151, may 31, 1986 (from version .s:s3ds:s3ds6150uu)
c     in subroutine findrad: calculate skn1 from rmassy instead of rmassx
c
c-----version s3ds6192, july 11, 1986 (from version .s:s3ds:s3ds6151uu)
c     change calculation of rotational kinetic energy to use z-rotational
c     velocity relative to the mass-averaged rotational velocity for cell.
c     also, change calculation of rotational kinetic energy for each
c     particle-size group to use z-rotational velocity relative to the
c     mass-averaged rotational velocity for that group.
c     also, make changes in boundary particle calculations.
c

c-----version s3ds6254, sept. 11, 1986 (from version .s:s3ds:s3ds6192uu)
c     allow fixed particles within cell (whole particles) and
c     correct cell dimensions for overlap of boundary particles and
c     overlap of fixed particles to obtain desired packing fraction.
c
c-----version s3ds6262, sept. 19, 1986 (from version .s:s3ds:s3ds6254uu)
c     implement monte-carlo estimation of net volumes for
c     boundary particles and other fixed particles (subroutine packing).
c     redefine deviatoric x-, y-, and z-translational velocities to
c     leave out gravitational part.
c
c-----version s3ds6295, oct. 22, 1986 (from version .s:s3ds:s3ds6262uu)
c     allow non-zero initial non-deviatoric velocity components in
c     x-, y-, and z-direction.
c
c-----version s3ds6321, nov. 17, 1986 (from version .s:s3ds:s3ds6295uu)
c     use deviatoric velocities to calculate group kinetic energies.
c     correct names to nybz0 and nybz1 in output file.
c     correct printout of fixed particles coordinates when icoord = 0.
c     correct initialization of rad for fixed particles.
c     do not allow epsy to be input as 0.
c
c-----version s3ds6334, nov. 30, 1986 (from version .s:s3ds:s3ds6321uu)
c     allow full cylindrical particles in z direction.
c     allow separate sknlb, elastb, and slopeb for interactions
c     between active particles and boundary or fixed particles.
c
c-----version s3ds6358, dec. 24, 1986 (from version .s:s3ds:s3ds6334uu)
c     corrections for nzcyl.gt.0
c
c-----verion s3ds7008, jan. 8, 1987 (from version .s:s3ds:s3ds6358uu)
c     new flag "ialt" to specify whether alternative k values (sknlb, skn2b
c     elastb, and slopeb are to apply to (0) active/boundary collisions
c     or (1) active/active collisions when both active particles are
c     not in first particle size group
c
c-----version s3ds7160, jun. 9, 1987 (from version .s:s3ds:s3ds7008uu)
c     allow planar boundaries on y and z faces of cell
c     specify nxby0=nzby0=1, or nybz0=nxbz0=1, etc.
c     added calculation of lower y-boundary particle forces
c
c-----version s3ds7177, jun 26,1987 from version .s:s3ds:s3ds7160
c     add velocity sqared drag force to subroutine integ1, read in drag
c     corrected initial orientation of fixed and boundary particles
c     fmub = separate friction coefficient for boundary or fixed particles
c     ixyz = flag to read coords, =0 for only fixed particles, =1 fixed & bound
c          =2 to read all particle coords.
c     allow negative radius for first zcylinder --  as outer bound
c
c-----version s3ds7257, sept 14, 1987 from version .o:3dshear:s3ds:s3ds7177
c     correct definition of frist z-cylinder (was sphere in s3ds7177)
c     correct alternate boundary friction in findrad initialization (fmub)
c     add edit of radii, masses, and moments of inertia before first time edit
c
c-----version s3ds6110. april 20, 1988 from version .o:3dshear:s3ds:s3ds7257uu
c     correct initialization of dx,dy,dz in subroutine init
c     add calculation of dx for boundary particles in subroutine integ2
c     add additional dummy zone for all particles with ystarty or ypystop
c        this changes mass assigned to zone 1 and nyzone from previous version
c     delete yshift correction when ireal.gt.0. in subroutine initstep
c     correct rotational kinetic energy diagnostic for zones (ie. index to ywz)
```

```
c      add vx,vy,vz,wx,wy,wz to inamelist input
c       use values read in according to flag ixyz:
c       ixyz=0,1,2 for fixed, fixed and boundary, or all particles, respectively
c       ixyz=3 for all particles - but x,y,z not to be scaled by cell size
c
c------version s3ds9335, nov 30, 1988 from version 3dshear/s3ds8110
c       add ialtk=-1 option to change friction, stiffness etc. to "b" values
c       after particle index ind2y1 (i.e. only for z-boundaries or cylinders)
c       variable ndx2 added to common
c
c------version s3dsnew1, dec 1988 from 3dshear/s3ds8335
c       sun unix version translated from 11nl octopus version
c       uses sun fortran with vms extension (includes namelist input)
c
c-----verion s3dsnew, mar. 24, 1989 from s3dsnew1
c     continue with sun unix version:
c     implement namelist input, begin conversion of dump coding,
c     clean-up some i/o problems, and re-arrange common.
c     code now will compile, but link list has not been corrected.
c
c-----version s3dsnewb, may 5, 1989 from s3dsnew
c       modified link-list to work with 32 or 64 bit integers along with
c       real*8 floating point numbers.  use idx for integers, jdx for real
c       modified dump so that only latest time is available (i.e., can
c       continue problem but not restart from earlier time.
c
c-----version s3ds91a.1000 May 18,1991
c     changed max dimension in s3dscmn to mp = 1000
c
c-----version 3ds91b.1k June 15, 1991
c     eliminated sign function calls & if tests in periodic image tests
c
c-----version 3ds91c.1k July 19, 1991 (from 3ds91b.1k)
c     minor changes to get thru IBM compiler:
c     removed data statements for variables in COMMON (for IBM risc6000)
c     corrected if-test in 'forces'  if ((i.and.j).gt.number(1))
c     type character variable in 'datasave', format statement in 'init'
c     'etime' function in 'datasave' modified (still incorrect on IBM)
c
c-----version 3ds91e.1k (from 3ds91c.1k)
c     added boundary x-velocity to vmaxx calculation in diagnost
c     removed redundant factor of two in vmaxx calculation in update
c     restored 'etime' in datasave
c
c-----3dshear version 2.0 (from 3ds91e.1k)
c     diagnostics are calculated and written to o3ds if nout or dtout
c     is nonzero.  x,y,z positions and x,y,z rotation angles are written
c     to o3dxq if noutx or dtoutx is nonzero.  velocities are written
c     to o3dv if noutv or dtoutv is nonzero.  dash1 and dash2 damping
c     factors were added.
c-----3dshear version 2.0b (from 2.0)
c     increased mp to 2500
c     completed logic for negative radius (i.e., inside) contacts
c        in subroutine forces (from 2-body code)
c     allowed rotating cylinders (i.e., set whz(i) = wz(i) for
c        zcylinders in init)
c     allowed override of additional variables in dumpread:
c        tzero, nzzero, dtoutx, dtoutv noutx, noutv
c
c-----3dshear version 2.01 (from 2.0b) May 1, 1993
c     versions 2.01a dimensioned 500, 2.01b dimensioned 3500
c      automatically place particles inside cylindrical boundary
c      call update based on displacement criteria instead of time
c
c-----3dshear version 2.02 (from 2.01) August 5, 1993
c     (2.02a dimensioned 500, 2.02b dimensioned 3500)
c     dash1, dash1b, dash2, dash2b damping factors implemented in forces
c     elast and elastb also apply to Hertz-like force models.
c     'slope' no longer used for damping in hertz force model.
c     eliminated inverse 12th power potential option in forces.
c     rearranged force routine to speed search thru linked-list.
c
c     (coding for damping forces not fully tested as of 8/5/93)
c     note: if extensive runs with 'slope' are planned then, skn2 should
c     be put back into contact linked-list for greater efficiency.
c
c     Sept 16, 1993 rearranged statements in diagnost and forces to
c          avoid warning errors from ibm f77 compiler.  Eliminated
c          redundent statements in forces, changed ranf to ranf1.
c
c----3dshear version 2.03 (from 2.02) February 9, 1993
c     added input variables wxzero, wyzero, wzzero for initial angular
c     velocities of the active particles (one fixed value for all
c     spheres, no random variation about mean).
c
c----3dshear version 2.04 (from 2.03) March 29, 1994
c     correct print time intervals in dumpread
c     eliminated divide by  zero in init (for cylindrical boundary)
c     added fmu and fmub as variables modified by restart file when
c     restarting from dump file (MUST INCLUDE THESE VARIABLES IN
c     RESTART FILE for them to be non-zero on restart).
c
c----3dshear version 2.05 (from 2.04) May 3, 1994
c     added input variable wzcyl both initially and on restart
c     (non-zero value overrides previous or alternate input values)
c     for the angular velocity of the 1st z-cylinder (boundary).
c     also, changed dump length of neighbor linked list.
c     files o3ds, o3dx, o3dv are closed after each write
c     CANNOT READ RESTART DUMP FILES FROM EARLIER VERSIONS!
c----version 2.05b
c     increased dimensions to allow 7100 particles, decreased
c     max yzones to 20, groups to 5, velocity groups to 10
c     changed orientation edit to quaternions in o3dxq file
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      include 's3dscmn'
c
c      data pi/3.1415926535897932/,ccpack/0.7404804/
c      data half/.5/,third/.3333333333333/,fourth/.25/,fifth/.2/
c      data sixth/.1666666666667/
c      data one/1./,two/2./,three/3./,four/4./,ten/10./
c      data trifle/1.e-50/,huge/1.e+50/
c      data mtl/1/, nebor/mp*0/

c     ieee = ieee_handler('set','invalid',sample_handler)
c     ieee = ieee_handler('set','division',sample_handler)

      version = '2.05b   '
      one = 1.
      two = 2.
      three = 3.
      four = 4.
```

```
      ten = 10.
      half = 0.5
      two5ths = four/ten
      threehf =three/two
      third = one/three
      fourth = 0.25
      fifth = 0.2
      sixth = one/6.
      ccpack = 0.7404804
      pi = four*atan(one)
      trifle = 1.e-50
      huge = 1.e+50
      mtl = 1
      do 1 i=1,mp
    1 nebor(i) = 0
c
c-----define machine dependent variables
      if(ivers.eq.2) then
c-----        use integer*4 in link-list (sun)
         i2or1 = 2
         ilor0 = 1
c-----        8 (byte) locations per floating pt. variable (sun)
         i8or1 = 8
      else
c-----        integers and reals same length (cray)
         i2or1 = 1
         ilor0 = 0
c-----        1 cray word per floating pt. variable
         i8or1 = 1
      endif
c
c-----open input and output files
      open(2,file='i3ds',status='old')
c
c-----set maximum index for linked list storage
      maxwd = 1 + nwd*mp*nnave
      imaxwd = i2or1*maxwd - ilor0
      lastll = nwd + nwd*mp*nnave
      ilastll = i2or1*lastll - ilor0
      llused = nwd
      illused = i2or1*llused - ilor0
c
c-----determine length of data to write to restart dump file.
      lenfp = 1 + ((loc(lastsc) - loc(firstsc))/i8or1)
      lenchr = 1 + ((loc(lastchr) - loc(oldtt1(1)))/i8or1)
c
c-----read input data
      call datain
c
      if(istart .ne. 0) then
c-----restart from dump of run istart
         call dumpread
c-----resume run
         goto 10
      endif
c
c-----initialize general parameters
      call init
c
c-----increase particle radii rad(i) until they reach radz(i)
      call findrad
      call update
c
 10   continue
c
c-----write header into to output files
      call o3dsout(0)
      if(iquat.eq.0) then
         call o3dxout(0)
      else
         call o3dxqout(0)
      endif
      call o3dvout(0)
c
c-----start integration
 20   continue
c
c-----update linked list of near neighbors if displacement to large
      delrup = delrup + dmax
      if(delrup.gt. half*search) call update
c
c-----initialize for integration step
      call initstep
c
c-----calculate interparticle forces
      call forces
c
c-----iterative integration of velocity equations to solve for vx, vy, and vz
c-----at start of current time step
      call integ1
c
c-----calculate diagnostics before completing the integration
      if(nout.gt.0) then
         call diagnost
c-----test for writing file o3ds (diagnostics)
         if(t.gt.tout+dtout-half*dt) call o3dsout(1)
      endif
c
c-----test for writing file o3dx (x,y,z,rotations) or o3dxq (x,y,z,quaternions)
      if(noutx.gt.0) then
         if(t.gt.toutx+dtoutx-half*dt) then
            if(iquat.eq.0) then
               call o3dxout(1)
            else
               call o3dxqout(1)
            endif
         endif
      endif
c
c-----test for writing file o3dv (velocities)
      if(noutv.gt.0) then
         if(t.gt.toutv+dtoutv-half*dt) call o3dvout(1)
      endif
c
c-----now finish this cycle of integration to obtain coordinates at time t + dt
c-----and estimation of velocities at time t + dt
      call integ2
c
c
c-----test for dumping
      if(t-dt .gt. tdump-half*dt) then
c-----increment dump time
```

```
        tdump = tdump + dtdump
c
c-----create/open dump file.
      open(4,file='d3ds',status='unknown',form='unformatted')
c
c-----write length of each data block in dump file (for consistency chk)
        write(4,iostat=icheck,err=998) lenchr,ilastll,lenfp,illused
     >  ,llused
c-----write characters to dumpfile
         write(4,iostat=icheck,err=998) (oldttl(i),i=1,lenchr)
c-----write linklist variables to dump file
        write(4,iostat=icheck,err=998) (ndx(i),i=1,illused)
        if(icheck.ne.0) goto 998
c-----write blank common to dump file
        write(4,iostat=icheck,err=998) (firstsc(i),i=1,lenfp)
        if(icheck.ne.0) goto 998
c
c-----close dumpfile after each time written
         close(4)
      endif
c
c
      if(itty.eq.1) then
c-----test for continue integration when in tty-interactive mode
        if(istop.ge.0) goto 20
      else
c-----test for continue integration when not in tty-interactive mode
        if(t.le.tmax+half*dt) goto 20
      endif
c
c
c-----mean collision frequency for entire run
      if((masst+mass).gt.0.) then
        colfrq = three*pack*sqrt((vsqmt+vsqm)/(masst+mass))/rave
      else
        colfrq = 0.
      endif
c
c-----mean estimated collisions per particle
      ncmax = tmax*colfrq
c
c-----mean counted collisions per particle
      nctot = two*totcol/np
c
      open(3,file='o3ds',status='old', access='append')
      write(3,301) tmax,colfrq,ncmax,nctot
      call o3dsout(2)
c
 999  if(ivers.eq.1) then
        call exit(1)
      else
        stop
      endif
c
 998  open(3,file='o3ds',status='old', access='append')
      write(3,398) icheck
      goto 999


c
c
 301  format(1p,///,
     & " tmax =   ",e11.4," $max time for run",/,
     & " colfrq = ",e11.4," $mean collision frequency",/,
     & " ncmax =   ",i5,5x," $estimated collisions per particle",/,
     & " nctot =   ",i5,5x," $counted collisions per particle",/,
     & " $$$")
c
 398  format(/,' error writing dump file, icheck = ',i9)
      end
c
c  ******    *****  *     *  *     *  ******
c  *     *  *     *  *     *  *  *  *  *     *
c  *     *  *     *  *     *  * * *  *  *     *
c  *****    *     *  *     *  *  *  *  *     *
c  *     *  *     *  *     *  *     *  *     *
c  *     *  *     *  *     *  *     *  *     *
c  ******    *****    *****   *     *  ******
c
      subroutine bound
      include 's3dscmn'
c
c-----this subroutine is called only if ireal.ne.0
c-----it assigns coord., vel., and other parameters for boundary particles
c
c-----index of y zone below zone 1
        nminus = 1
c-----index of y zone above zone nyzone
        nplus = nyzone
c-----multiplier for calculating shear rate in boundary y zones,
        yfact = 1.
c
      packy0 = 0.
      if(nby0.eq.1) then
c-----plane
        ip = indly0
        rad(ip) = radz(ip)
        x(ip) = 0.
        y(ip) = -rad(ip)
        z(ip) = 0.
        vx(ip) = vxby0
        vy(ip) = vyby0
        vz(ip) = vzby0
c
      elseif(nby0.gt.1) then
c-----expanded, hexagonal-close-packed coord. for xz boundary at y = zero
        ip = indly0 - 1
      do 10 j=1,nzby0
        zz = (j-1)*zcell/nzby0
        xx = (1 - mod(j,2))*half*xcell/nxby0
      do 10 i=1,nxby0
        ip = ip + 1
        x(ip) = xx + (i-1)*xcell/nxby0
        z(ip) = zz
        y(ip) = 0.
        vx(ip) = vxby0
        vy(ip) = vyby0
        vz(ip) = vzby0
        wx(ip) = 0.
        wy(ip) = 0.
        wz(ip) = 0.
        rad(ip) = radz(ip)
        npos(ip) = 1.
```

```
      mneg(ip) = 0.
      rpos(ip) = 1.
      rneg(ip) = 0.
c-----packing contribution from half-particles at y = zero
      packy0 = packy0 + half*vol(ip)
 10   continue
c-----correct for overlap of boundary half-particles at y = zero
c-----no overlap is assumed with particles on the xy boundary, if any.
      do 110 i=ind1y0,ind2y0-1
      do 110 j=i+1,ind2y0
         dxij = min(abs(x(j)-x(i))
     1              ,abs(x(j)-x(i)-xcell)
     1              ,abs(x(j)-x(i)+xcell))
         dzij = min(abs(z(j)-z(i))
     1              ,abs(z(j)-z(i)-zcell)
     1              ,abs(z(j)-z(i)+zcell))
         dijsq = dxij*dxij + dzij*dzij
         if(dijsq.ge.(radz(i) + radz(j))**2) goto 110
         dij = sqrt(dijsq)
         hi = (radz(j)**2 - (dij - radz(i))**2)/(2.*dij)
         hj = (radz(i)**2 - (dij - radz(j))**2)/(2.*dij)
         packy0 = packy0 - sixth*pi
     1                   *(hi*hi*(3.*radz(i) - hi)
     2                   + hj*hj*(3.*radz(j) - hj))
 110  continue
      endif
c
c
      packy1 = 0.
      if(nby1.eq.1) then
c-----plane
        ip = ind1y1
        rad(ip)= radz(ip)
        x(ip) = 0.
        y(ip) = ycell + rad(ip)
        z(ip) = 0.
        vx(ip) = vxby1
        vy(ip) = vyby1
        vz(ip) = vzby1
c
      elseif(nby1.gt.1) then
c-----expanded, hexagonal-close-packed coord. for xz boundary at y = ycell
        ip = ind1y1 - 1
      do 20 j=1,nzby1
         zz = (j-1)*zcell/nzby1
         xx = (1 - mod(j,2))*half*xcell/nxby1
      do 20 i=1,nxby1
         ip = ip + 1
         x(ip) = xx + (i-1)*xcell/nxby1
         z(ip) = zz
         y(ip) = ycell
         vx(ip) = vxby1
         vy(ip) = vyby1
         vz(ip) = vzby1
         wx(ip) = 0.
         wy(ip) = 0.
         wz(ip) = 0.
         rad(ip)= radz(ip)
         npos(ip) = 0
         mneg(ip) = nyzone
         rpos(ip) = 0.
         rneg(ip) = 1.
         packy1 = packy1 + half*vol(ip)
 20   continue
c-----correct for overlap of boundary half-particles at y = ycell
c-----no overlap is assumed with particles on the xy boundary, if any.
      do 120 i=ind1y1,ind2y1-1
      do 120 j=i+1,ind2y1
         dxij = min(abs(x(j)-x(i))
     1              ,abs(x(j)-x(i)-xcell)
     1              ,abs(x(j)-x(i)+xcell))
         dzij = min(abs(z(j)-z(i))
     1              ,abs(z(j)-z(i)-zcell)
     1              ,abs(z(j)-z(i)+zcell))
         dijsq = dxij*dxij + dzij*dzij
         if(dijsq.ge.(radz(i) + radz(j))**2) goto 120
         dij = sqrt(dijsq)
         hi = (radz(j)**2 - (dij - radz(i))**2)/(2.*dij)
         hj = (radz(i)**2 - (dij - radz(j))**2)/(2.*dij)
         packy1 = packy1 - sixth*pi
     1                   *(hi*hi*(3.*radz(i) - hi)
     2                   + hj*hj*(3.*radz(j) - hj))
 120  continue
c
      endif
c
c
      packz0 = 0.
      if(nbz0.eq.1) then
c-----plane
        ip = ind1z0
        rad(ip) = radz(ip)
        x(ip) = 0.
        y(ip) = 0.
        z(ip) = -rad(ip)
c
      elseif(nbz0.gt.1) then
c-----xy-boundary at z = zero
c**********************************************************************
c**********************************************************************
c-----generate coordinates, similar to xz boundary
c**********************************************************************
c**********************************************************************
c-----correct for overlap of boundary half-particles at z = zero
c-----no overlap is assumed with particles on the xz boundary, if any.
      do 130 i=ind1z0,ind2z0-1
      do 130 j=i+1,ind2z0
         dxij = min(abs(x(j)-x(i))
     1              ,abs(x(j)-x(i)-xcell)
     1              ,abs(x(j)-x(i)+xcell))
         dyij = min(abs(y(j)-y(i))
     1              ,abs(y(j)-y(i)-ycell)
     1              ,abs(y(j)-y(i)+ycell))
         dijsq = dxij*dxij + dyij*dyij
         if(dijsq.ge.(radz(i) + radz(j))**2) goto 130
         dij = sqrt(dijsq)
         hi = (radz(j)**2 - (dij - radz(i))**2)/(2.*dij)
         hj = (radz(i)**2 - (dij - radz(j))**2)/(2.*dij)
         packz0 = packz0 - sixth*pi
     1                   *(hi*hi*(3.*radz(i) - hi)
     2                   + hj*hj*(3.*radz(j) - hj))
 130  continue
```

```
      endif
c
c
      packz1 = 0.
      if(nbz1.eq.1) then
c-----plane
        ip = indlz1
        rad(ip) = radz(ip)
        x(ip) = 0.
        y(ip) = 0.
        z(ip) = zcell + rad(ip)
c
      elseif(nbz1.gt.1) then
c-----xy-boundary at z = zcell
c**********************************************************************
c**********************************************************************
c-----generate coordinates, similar to xz boundary
c**********************************************************************
c**********************************************************************
c-----correct for overlap of boundary half-particles at z = zcell
c-----no overlap is assumed with particles on the xz boundary, if any.
      do 140 i=indlz1,ind2z1-1
      do 140 j=i+1,ind2z1
         dxij = min(abs(x(j)-x(i))
     1              ,abs(x(j)-x(i)-xcell)
     1              ,abs(x(j)-x(i)+xcell))
         dyij = min(abs(y(j)-y(i))
     1              ,abs(y(j)-y(i)-ycell)
     1              ,abs(y(j)-y(i)+ycell))
         dijsq = dxij*dxij + dyij*dyij
         if(dijsq.ge.(radz(i) + radz(j))**2) goto 140
         dij = sqrt(dijsq)
         hi = (radz(j)**2 - (dij - radz(i))**2)/(2.*dij)
         hj = (radz(i)**2 - (dij - radz(j))**2)/(2.*dij)
         packz1 = packz1 - sixth*pi
     1                   *(hi*hi*(3.*radz(i) - hi)
     2                   + hj*hj*(3.*radz(j) - hj))
 140  continue
      endif
c
      return
      end
c
c ******     ***    *******   ***    ***  *     *
c *     *     *     *     *     *    *  *   **   *
c *     *     *     *     *     *    *  *  * *   *
c *     *     *     *******     *    *  *  *  *  *
c *     *     *     *     *     *    *  *  *   * *
c *     *     *     *     *     *    *  *  *    **
c ******    ***    *     *    ***  ***  *  *     *
c
      subroutine datain
      include 's3dscrn'
      character*80 chrdum
c
      namelist /var/ np,nxby0,nzby0,nxby1,nzby1,nxbz0,nybz0
     + ,nxbz1,nybz1,nfix,nfxpl,nzcyl,ncmax,nout,noutx,noutv
     + ,nczero,ntcol,nvel,ndump,nyzone,ireal,imirr,iterva
     1 ,izeta,icoord,iquat,itty,ihertz,istart,ialtk
     2 ,tmax,dt,dtout,dtoutx,dtoutv,dtdump,tzero,edot,epsv,pack
     2 ,vave,vxzero,vyzero,vzzero,vseed,wxzero,wyzero,wzzero,wzcyl
     3 ,sknl,elast,slope,ratk,fmu,power,tstart,rmassz
     4 ,xcell,ycell,zcell,xyrat,zyrat,gravx,gravy,gravz
     5 ,vxby0,vxby1,vyby0,vyby1,vzby0,vzby1
     6 ,vxbz0,vxbz1,vybz0,vybz1,vzbz0,vzbz1,q1old,q2old,q3old,q4old
     7 ,search,xmax,draddt,ystart,ystop
     8 ,radz,number,radius,pmass,planex,planez
     8 ,x,y,z,sknlb,elastb,slopeb
     9 ,drag,ixyz,fmub,vx,vy,vz,wx,wy,wz,dash1,dash2,dash1b,dash2b
     $ ,finis
c
c**********************************************************************
c-----input title
      read (2,201) (title(i),i=1,10)
c
c-----find run number given on title line (the 3 digits after "i3ds")
      write(chrdum,'(10a8)') (title(i),i=1,10)
      read(chrdum,'(80a1)') (nchar(i),i=1,80)
      i = 1
      nnrun = 0
 1    continue
      if(nchar(i).eq."i".and.nchar(i+1).eq."3".and.nchar(i+2).eq."d"
     1     .and.nchar(i+3).eq."s") then
        write(chrdum,'(3a1)') (nchar(i+j),j=4,6)
        read(chrdum,'(i3)') nnrun
      else
        i = i + 1
        if(i.lt.70) goto 1
      endif


c
c**********************************************************************
c
c-----initialize control parameters
c-----np =      total number of boundary and non-boundary particles
      np = 125
c
c-----nxby0 = number of boundary particles in x direction at y = zero
      nxby0 = 0
c
c-----nzby0 = number of boundary particles in z direction at y = zero
c             (nzby0 must be an even number)
      nzby0 = 0
c
c-----nxby1 = number of boundary particles in x direction at y = ycell
      nxby1 = 0
c
c-----nzby1 = number of boundary particles in z direction at y = ycell
c             (nzby1 must be an even number)
      nzby1 = 0
c
c-----nxbz0 = number of boundary particles in x direction at z = zero
      nxbz0 = 0
c
c-----nybz0 = number of boundary particles in z direction at z = zero
c             (nybz0 must be an even number)
      nybz0 = 0
c
c-----nxbz1 = number of boundary particles in x direction at z = zcell
      nxbz1 = 0
c
c-----nybz1 = number of boundary particles in z direction at z = zcell
```

```
c             (nybz1 must be an even number)
      nybz1 = 0
c
c-----nfix =    other fixed particles
      nfix = 0
c
c-----nfxpl = number of fixed planes
c             the orientation determined by the quaternions
c             the extents determined by xcell and zcell
      nfxpl = 0

c-----nzcyl =  number of cylinders parallel to z axis
c             if nzcyl is (1) and rad(indlcz) is negative then assume
c             the z-cylinder is the outer boundary for problem
      nzcyl = 0
c
c-----ncmax =  number of collisions per particle during entire run (est.).
c             note: this is used only if input tmax.eq.0.
      ncmax = 50
c
c-----nout =    number of times to write file o3ds (usual 3dshear output)
c             note: this is used only if dtout.eq.0.
      nout = 25
c
c-----noutx =  number of times to write file o3dxq (x,y,z,quaternions)
c             note: this is used only if dtoutx.eq.0.
      noutx = 100
c
c-----noutv =  number of times to write file o3dv (velocities)
c             note: this is used only if dtoutv.eq.0.
      noutv = 100
c
c-----nczero =  number of collisions per particle before restart cum. ave.
c             note: this is used only if input tzero.eq.0.
      nczero = 5
c
c-----ntcol =  number of time steps during a collision
c             note: ntcol and elast are used for calculating the
c             time step, dt, only if the input dt.eq.0.
      ntcol = 50
c
c-----nvel =    number of intervals for saving velocity distribution
      nvel = nvelmx
c
c-----ndump =  number of times to write restart dump.
c             note: this is used only if dtdump .eq. 0.
c     ndump = 10
c
c-----nyzone =  number of zones in y direction for diagnostics
c             note: maximum nyzone allowed is myzone
      nyzone = 5
c
c-----ireal =  if ireal.eq.0 the all boundaries are periodic
c             if ireal.eq.1 then planes normal to y axis are real
c             if ireal.eq.2 then planes normal to y axis are real and
c                           planes normal to z axis are also real
c               note: if ireal.ne.0 then edot must be zero.
      ireal = 0
c
c-----imirr =    if imirr.eq.1 & ireal.gt.0 then plane y=ycell is a mirror
      imirr = 0
c
c-----itermv =  maximum number of iterations for velocity integration
c             (itermv must be .ge. 1)
      itermv = 3
c
c-----izeta =  if izeta.eq.0, do not keep trans. kinetic energy constant
c             if izeta.eq.1, use hoover zeta method to keep constant t
c             if izeta.eq.2, rescale dev. vel. to keep constant t
      izeta = 0
c
c-----icoord =  if icoord.eq.0, write out coordinates in primary cell
c             if icoord.ne.0, write out coordinates in present cell
      icoord = 0
c
c-----iquat =   if iquat.eq.0, write out rotations as successive angles (o3dx)
c             if iquat.ne.0, write out rotations as quaternions (o3dxq)
      iquat = 0
c
c-----itty =    if itty.eq.1 program can be terminated from tty
      itty = 0
c
c-----ihertz =  if ihertz.eq.1 use hertz 3/2 power law force
c             if ihertz.eq.0 use latching spring force law
      ihertz = 0
c
c-----istart =  flag to determine whether to restart run.
c             if istart is nonzero, then restart run using values
c             from dump file.
      istart = 0
c
c-----ialtk =   flag to specify whether to use skn1b, skn2b, slopeb
c             and elastb for (0) collisions between an active particle
c             and boundary, or (1) collisions between two active
c             particles when both are outside of first size group.
c             (-1) use alternate values only for z-boundaries, fixed
c             particles and cylinders (i.e., let x- and y-boundary
c             particles have same values as active particles).

      ialtk = 0
c-----ixyz = flag to read in coordinates and velocities for particles
c           = 0 only use coords and velocities of fixed particles
c           = 1 use x,y,z,vx,vy,vz,wx,wy,wz for boundary and fixed particles
c           = 2 use coords & velocities  for all particles
c           = 3 use for all particles but do not scale by cell size
      ixyz = 0
c
c
c-----tmax =    maximum time for entire run
c             note: if tmax.eq.0. then tmax is determined by ncmax
      tmax = 1.
c
c-----dt =      time step
c             note: if dt.eq.0. then dt is determined by ntcol
      dt = 0.
c
c-----dtout =   time interval for writing file o3ds (diagnostics)
c             note: if dtout.eq.0. then dtout is determined by nout
      dtout = 0.
c
c-----dtoutx =  time interval for writing file o3dxq (x,y,z,quaternions)
c             note: if dtoutx.eq.0. then dtoutx is determined by noutx
```

```
      dtoutx = 0.
c
c-----dtoutv =  time interval for writing file oldv (velocities)
c               note: if dtoutv.eq.0. then dtoutv is determined by noutv
      dtoutv = 0.
c
c-----dtdump =  time interval for writing to dump file.
c               note: if dtdump .eq. 0. then dtdump is determined by ndump
c     dtdump = 0.
c
c-----tzero =   time at which cumulative averaging is restarted
c               note: if tzero.eq.0. then tzero is determined by nczero
      tzero = 0.
c
c-----edot =    strain rate
      edot = 0.
c
c-----epsv =    relative error tolerance for determining packing fraction
c               when real boundary particles are used.
      epsv = 0.00001
c
c-----xcell =   cell length in x dir.
      xcell = 0.
c
c-----ycell =   cell length in y dir.
      ycell = 0.
c
c-----zcell =   cell length in z dir.
      zcell = 0.
c
c-----xyrat =   ratio of x cell length to y cell length
      xyrat = 1.
c
c-----zyrat =   ratio of z cell length to y cell length
      zyrat = 1.
c
c-----pack =    packing fraction (must be greater than zero)
c               note: if code adds more boundary particles,
c               then pack will be modified from input value
      pack = 0.7
c
c-----vave =    average initial velocity relative to shear field
c               (if vave.eq.0., do not normalize initial velocities).
      vave = 1.
c
c-----vxzero =  initial non-deviatoric transl. vel. in x-dir.
      vxzero = 0.
c
c-----vyzero =  initial non-deviatoric transl. vel. in y-dir.
      vyzero = 0.
c
c-----vzzero =  initial non-deviatoric transl. vel. in z-dir.
      vzzero = 0.
c
c-----initialize angular velocities for active particles (ixyz < 2)
c-----wxzero =  initial angular vel. about x-axis
      wxzero = 0.
c
c-----wyzero =  initial angular vel. about y-axis
      wyzero = 0.
c
c-----wzzero =  initial angular vel. about z-axis
      wzzero = 0.
c
c-----wzcycl =  angular velocity of 1st zcylinder (boundary)
c               if non-zero, initiali or on restart, will override
c               wz(i) value
      wzcyl = 0.
c
c-----vseed =   seed (between 0. and 1.) for generating a new
c               sequence of random  initial particle velocities
      vseed = 0.
c
c-----skn1 =    normal force coefficient for loading
      skn1 = 1.e+06
c
c-----elast =   coefficient of restitution
c               if slope is nonzero, elast is not used for
c               determining normal force coefficient for unloading;
c               but an approximate value of elast must still be input
c               for use in calculating the time step, when the input
c               value of dt is zero.
      elast = 0.8
c
c-----slope =   if slope is nonzero, use alternative method for
c               determining normal force coefficient for unloading
      slope = 0.
c
c-----loading and unloading coefficients for collisions between
c     active particles and boundary or other fixed particles.
c     assign default values of -1.; then after namelist input if they are
c     still -1. let sknlb = skn1, elastb = elast, and slopeb = slope :
c      same for fmub, if = -1. set fmub = fmu
c
c-----sknlb =   normal force coefficient for loading
      sknlb = -1.
c
c-----elastb =  coefficient of restitution
      elastb = -1.
c
c-----slopeb = alternative parameter for unloading
      slopeb = -1.
c
c-----dash1    linear viscous damping coefficient
      dash1 = 0.
c
c-----dash2    overlap (i.e., 'a') weighted damping coefficient
      dash2 = 0.
c
c-----dash1b = alternative linear damping coef.
      dash1b = -1.
c
c-----dash2b = alternative overlap-weighted damping coef.
      dash2b = -1.
c
c-----ratk =    ratio skt0/skn for tangential force calculation
      ratk = 1.
c
c-----fmu =     friction coefficient
      fmu = 0.
c
c-----fmub =    alternative friction coefficient
```

```
      fmub = -1.
c
c-----power =  exponent for tangential force calculation
      power = 0.
c
c-----rmassz =  mass of particle having unit radius
      rmassz = 1.
c
c-----tstart =  time at which to restart run.
c              if tstart is zero,use final dump when restarting.
c     tstart = 0.
c
c
c-----gravx =  acceleration of gravity in the x direction
      gravx = 0.
c
c-----gravy =  acceleration of gravity in the y direction
      gravy = 0.
c
c-----gravz =  acceleration of gravity in the z direction
      gravz = 0.
c
c-----vxby0 =  velocity in x direction of real boundary at y = zero
c       note: vxby0 is used only if ireal.gt.0 & nby0.gt.0 & ixyz.lt.1
      vxby0 = 0.
c
c-----vxby1 =  velocity in x direction of real boundary at y = ycell
c       note: vxby1 is used only if ireal.gt.0 & nby1.gt.0 & ixyz.lt.1
      vxby1 = 0.
c
c-----vyby0 =  velocity in y direction of real boundary at y = zero
c       note: vyby0 is used only if ireal.gt.0 & nby0.gt.0 & ixyz.lt.1
      vyby0 = 0.
c
c-----vyby1 =  velocity in y direction of real boundary at y = ycell
c       note: vyby1 is used only if ireal.gt.0 & nby1.gt.0 & ixyz.lt.1
      vyby1 = 0.
c
c-----vzby0 =  velocity in z direction of real boundary at y = zero
c       note: vzby0 is used only if ireal.gt.0 & nby0.gt.0 & ixyz.lt.1
      vzby0 = 0.
c
c-----vzby1 =  velocity in z direction of real boundary at y = ycell
c       note: vzby1 is used only if ireal.gt.0 & nby1.gt.0 & ixyz.lt.1
      vzby1 = 0.
c
c-----vxbz0 =  velocity in x direction of real boundary at z = zero
c       note: vxbz0 is used only if ireal.gt.0 & nbz0.gt.0 & ixyz.lt.1
      vxbz0 = 0.
c
c-----vxbz1 =  velocity in x direction of real boundary at z = zcell
c       note: vxbz1 is used only if ireal.gt.0 & nbz1.gt.0 & ixyz.lt.1
      vxbz1 = 0.
c
c-----vybz0 =  velocity in y direction of real boundary at z = zero
c       note: vybz0 is used only if ireal.gt.0 & nbz0.gt.0 & ixyz.lt.1
      vybz0 = 0.
c
c-----vybz1 =  velocity in y direction of real boundary at z = zcell
c       note: vybz1 is used only if ireal.gt.0 & nbz1.gt.0 & ixyz.lt.1
      vybz1 = 0.
c
c-----vzbz0 =  velocity in z direction of real boundary at z = zero
c       note: vzbz0 is used only if ireal.gt.0 & nbz0.gt.0 & ixyz.lt.1
      vzbz0 = 0.
c
c-----vzbz1 =  velocity in z direction of real boundary at z = zcell
c       note: vzbz1 is used only if ireal.gt.0 & nbz1.gt.0 & ixyz.lt.1
      vzbz1 = 0.
c
c-----search =  distance for near-neighbor search
      search = 0.
c
c-----xmax =  maximum allowable relative x-coordinate
c             if abs(x).gt.xmax*xcell,
c             then move all particles back to primary cell
      xmax = 5.
c
c-----raddt = rate of increase of particle radii from rad(i) to radz(i)
c             after rad(i) have been set to their maximum values at t = 0.
      raddt = 1.
c
c-----ystart = y-coordinate of bottom of first y zone
      ystart = 0.
c
c-----ystop  = y-coordinate of top of last y zone
c-----note: if ystop = -1., then ystop will be set to ycell
c-----as soon as the value of ycell has been determined.
      ystop  = -1.
c
c-----radz(i) =  radius of ith particle
      do 5 i=1,mp
        radz(i) = 1.
 5    continue
c
c-----alternate method of inputing particle size and mass data
      do 10 i=1,mgroup
c-----number of particles having radius=radius(i) and mass=pmass(i)
c-----Added support for dimensioning arbitrary fixed planes
        number(i) = 0
        radius(i) = 0.
        pmass(i)  = 0.
        planex(i) = 0.
        planez(i) = 0.
 10   continue
c
c-----finis =  if finis is nonzero, then namelist input is terminated
      finis = 0.
c
c**********************************************************************
c**********************************************************************
c-----input control parameters to over-ride above values
 20   continue
      read (unit=2,nml=var,err=997,end=25)
      if(finis.eq.0.) goto 20
c**********************************************************************
c**********************************************************************
c
c-----error checks
 25   if((nrun.gt.0).and.(nrun.ne.nrrun)) goto 998
      if(epsv.le.0.) epsv = 1.e-06
      if(nyzone.gt.(myzone-1))nyzone = myzone - 1
```

```
c
      return
c
c-----error exits
 999  call exit(1)
c
 998  write(3,398)
      goto 999
 997  write(3,397)
      go to 999
c
 201  format(10a8)
 397  format(/," error reading namelist input file, unit 2")
 398  format(//," run number inconsistency")
 601  format(10a8)
 602  format(80a1)
 603  format(3a1)
 604  format(i3)
c
      end
c
c  ******   *******  *       ********  *******  *******  *     *
c   *    *  *        *       *            *     *        **   **
c   *    *  *        *       *            *     *        * * * *
c   *    *  ****     *       ****         *     ****     *  *  *
c   *    *  *        *       *            *     *        *     *
c   *    *  *        *       *            *     *        *     *
c  ******   *******  *******  ********     *     *******  *     *
c
      subroutine deletem
      include 's3dscmn'
      real*8 rot(3,3)
c
c-----this subroutine loops through all near neighbors in the linked-
c      list and deletes contacts that are beyond the maximum distance.
c      it is only used when the maximum distance has been reduced to
c      save total memory used for near-neighbor storage.
c       jdx is pointer for real nos.,  idx for integers
c
c-----examine all near-neighbor pairs in linked list
      do 100 i=imin,imax
        idx1 = nebor(i)
        if(idx1.eq.0) goto 100
         idx1 = i1or1*idx1 - i1or0
        jdx = idx1
         idx = idx1
c
c-----    loop thru the i-th particle's linked list
   10   j = ndx(idx)
        if(j.eq.0) goto 100
c
        rsum = rad(i) + rad(j)
c
c-----delta x and delta y
        rx = x(j) - x(i)
        ry = y(j) - y(i)
        rz = z(j) - z(i)
c
c-----cylinders
        if(j.ge.indlcz) rz = 0.
c-----boundary planes
        if(((j.eq.indly0).and.(nby0.eq.1)).or.
     1     ((j.eq.indly1).and.(nby1.eq.1))) then
          rx = 0.
          rz = 0.
        elseif(((j.eq.indlz0).and.(nbz0.eq.1)).or.
     1         ((j.eq.indlz1).and.(nbz1.eq.1))) then
          rx = 0.
          ry = 0.
        endif
c
      if(ireal.eq.0) then
c-----find nearest image delta-y
        inty = (ry + ycell1h)*ycell1 + 10.
        yshift = inty - 10
        ry = ry - yshift*ycell
c-----shear correction to delta x
        rx = rx - yshift*(edot*tshift - int(edot*tshift))
      endif
c
c-----find nearest image delta x
        intx = (rx + xcell1h)*xcell1 + 10.
        intx = intx - 10
        rx = rx -intx*xcell
c
      if(ireal.lt.2) then
c-----find nearest image delta z
        intz = (rz + zcell1h)*zcell1 + 10.
        intz = intz - 10
        rz = rz - intz*zcell
      endif
c
c-----fixed planes
      if((j.ge.ind1fxp).and.(j.le.ind2fxp)) then
          xplanh = xplnh(j)
          zplanh = zplnh(j)
c-----rotate coordinate system to make plane horizontal
        call matrot(q1old(j),q2old(j),q3old(j),-q4old(j),rot)
        rxr = rx*rot(1,1) + ry*rot(1,2) + rz*rot(1,3)
        ryr = rx*rot(2,1) + ry*rot(2,2) + rz*rot(2,3)
        rzr = rx*rot(3,1) + ry*rot(3,2) + rz*rot(3,3)
c-----detect borders of plane
        if(abs(rxr).gt.xplanh) then
          rxr = rxr - sign(xplanh,rxr)
        else
          rxr = 0
        endif
        if(abs(rzr).gt.zplanh) then
          rzr = rzr - sign(zplanh,rzr)
        else
          rzr = 0
        endif
c-----rotate back the coordinate system
        call matrot(q1old(j),q2old(j),q3old(j),q4old(j),rot)
        rx = rxr*rot(1,1) + ryr*rot(1,2) + rzr*rot(1,3)
        ry = rxr*rot(2,1) + ryr*rot(2,2) + rzr*rot(2,3)
        rz = rxr*rot(3,1) + ryr*rot(3,2) + rzr*rot(3,3)
      endif
c
        rijsq = rx*rx + ry*ry + rz*rz
        rnear2 =(rsum + search)**2
c
```

```
c****       if(rijsq.lt.rnear2) go to 20
      if(((rad(j).gt.0.).and.(rijsq.lt.rnear2)).or.
     >   ((rad(j).lt.0.).and.(rijsq.gt.rnear2))) go to 20
c-------------------------------------------------------------
c-----delete entry in linked list of i
        if(nebor(i).ne.jdx) goto 15
c
c       this is first entry in i's linked list
          nebor(i)= next(idx)
          next(idx) = mtl
          ndx(idx) = 0
          mtl= jdx
          jdxl = nebor(i)
           idxl = i2or1*jdxl - i1or0
          jdx = jdxl
           idx = idxl
        if(jdx.eq.0) goto 100
        goto 10
c
c       this is not first entry in i's linked list
   15     nxtl = next(idx)
          next(idx) = mtl
          ndx(idx) = 0
          mtl = jdx
          jdx = nxtl
           idx = i2or1*jdx - i1or0
          next(idxl) = nxtl
        if(jdx.eq.0) goto 100
        goto 10
c-------------------------------------------------
   20   continue
        jdxl = jdx
        idxl = idx
        jdx = next(idx)
        idx = i2or1*jdx - i1or0
        if(jdx.ne.0) goto 10
  100 continue
        return
        end
c
c
c ******   ***    ***     ******  *     *   *****    ******  *******
c *     *   *     *   *    *     *  *     *  *     *  *     *     *
c *     *   *    *     *   *     *  *     *  *        *     *     *
c *     *   *    *     *   *     *  *     *   *****   ******      *
c *     *   *    *     *   *     *  *     *        *  *           *
c *     *   *     *   *    *     *   *   *   *     *  *           *
c ******   ***     ***     ******    ***     *****   *           *
c
      subroutine diagnost
      include 'a3ddcmn'
c-----calculate diagnostics
c
c-----mass average z-rotational velocity and x-, y-, and z-translational
c-----velocities relative to shear for non-boundary particles in cell
        wzsum = 0.
        vxsum = 0.
        vysum = 0.
        vzsum = 0.
      do 3 i=ind1,ind2
        wzsum = wzsum + rmass(i)*wz(i)
        vxsum = vxsum + rmass(i)*(vx(i) - edot*y(i))
        vysum = vysum + rmass(i)*vy(i)
        vzsum = vzsum + rmass(i)*vz(i)
 3    continue
        wzave = wzsum/dnass
        vxave = vxsum/dnass
        vyave = vysum/dnass
        vzave = vzsum/dnass
c
c-----mass average z-rotational velocity and x-, y-, and z-translational
c-----velocities relative to shear for non-boundary particles in cell
c-----in each size group
        ip = 0
      do 7 j=1,ngroup
        gwzsum = 0.
        gdnass = 0.
        gvxsum = 0.
        gvysum = 0.
        gvzsum = 0.
      do 5 i=1,number(j)
        ip = ip + 1
        gdnass = gdnass + rmass(ip)
        if(ip.gt.ind2) goto 5
        gwzsum = gwzsum + rmass(ip)*wz(ip)
        gvxsum = gvxsum + rmass(ip)*(vx(ip) - edot*y(ip))
        gvysum = gvysum + rmass(ip)*vy(ip)
        gvzsum = gvzsum + rmass(ip)*vz(ip)
 5    continue
        gwzave(j) = gwzsum/gdnass
        gvxave(j) = gvxsum/gdnass
        gvyave(j) = gvysum/gdnass
        gvzave(j) = gvzsum/gdnass
 7    continue
c
        ip = 0
      do 9 j=1,ngroup
      do 8 i=1,number(j)
        ip = ip + 1
        if(ip.gt.ind2) goto 8
c-----x-velocity distribution for free particles in particle size group j
        vdevx = vx(ip) - edot*y(ip) - gvxave(j)
        if(vdevx.lt.0.) then
          n = nvelhalf + int(vdevx*delvxi)
          if(n.lt.0) n = 1
        else
          n = nvelhalf + int(vdevx*delvxi) + 1
          if(n.gt.nvel) n = nvel
        endif
        gbinvx(n,j) = gbinvx(n,j) + 1.
c-----y-velocity distribution for free particles in particle size group j
        vdevy = vy(ip) - gvyave(j)
        if(vdevy.lt.0.) then
          n = nvelhalf + int(vdevy*delvyi)
          if(n.lt.0) n = 1
        else
          n = nvelhalf + int(vdevy*delvyi) + 1
          if(n.gt.nvel) n = nvel
        endif
        gbinvy(n,j) = gbinvy(n,j) + 1.
c-----z-velocity distribution for free particles in particle size group j
        vdevz = vz(ip) - gvzave(j)
```

```
            if(vdevz.lt.0.) then
              n = nvelhalf + int(vdevz*delvzi)
              if(n.lt.0) n = 1
            else
              n = nvelhalf + int(vdevz*delvzi) + 1
              if(n.gt.nvel) n = nvel
            endif
            gbinvz(n,j) = gbinvz(n,j) + 1.
c-----special cell diagnostics for multi-size distributions
            dgekin(j) = dgekin(j) + rmass(ip)*(vdevx**2
     1        + vdevy**2 + vdevz**2)
            dgerot(j) = dgerot(j) + rmnt(ip)*(wx(ip)**2 + wy(ip)**2
     1        + (wz(ip) - gwzave(j))**2)
            dgflox(j) = dgflox(j) + (vx(ip) - edot*y(ip))*rmass(ip)
            dgfloy(j) = dgfloy(j) + vy(ip)*rmass(ip)
            dgfloz(j) = dgfloz(j) + vz(ip)*rmass(ip)
 8        continue
            gekin(j) = gekin(j) + half*dgekin(j)*vcelli
            gerot(j) = gerot(j) + half*dgerot(j)*vcelli
            gflox(j) = gflox(j) + dgflox(j)*vcelli
            gfloy(j) = gfloy(j) + dgfloy(j)*vcelli
            gfloz(j) = gfloz(j) + dgfloz(j)*vcelli
 9        continue
c
          imove = 0
          do 10 i=ind1,ind2
c-----deviatoric x-, y-, and z-translational velocities in cell
          vdevx = vx(i) - edot*y(i) - vxave
          vdevy = vy(i) - vyave
          vdevz = vz(i) - vzave
c
c-----particle-mass-weighted velocity square
          vsq = vdevx**2 + vdevy**2 + vdevz**2
          vsq = rmass(i)*vsq
c
c-----sum of particle-mass-weighted velocity squares
          dvsqn = dvsqn + vsq
c
c-----deviatoric translational kinetic energy
          dekin = dekin + half*vsq
c
c-----kinetic contribution to stress tensor
          dpxxk = dpxxk + rmass(i)*vdevx*vdevx
          dpxyk = dpxyk + rmass(i)*vdevx*vdevy
          dpxzk = dpxzk + rmass(i)*vdevx*vdevz
          dpyyk = dpyyk + rmass(i)*vdevy*vdevy
          dpyzk = dpyzk + rmass(i)*vdevy*vdevz
          dpzzk = dpzzk + rmass(i)*vdevz*vdevz
c
c-----rotational kinetic energy (using wz relative to mean shear rate)
          derot = derot + half*rmnt(i)*(wx(i)**2
         1   + wy(i)**2 + (wz(i) - wzave)**2)
c
c-----sum of mass-weighted spin angular momenta (using full wz)
          dspnx = dspnx + rmass(i)*rmnt(i)*wx(i)
          dspny = dspny + rmass(i)*rmnt(i)*wy(i)
          dspnz = dspnz + rmass(i)*rmnt(i)*wz(i)
c
c-----save x-velocity distribution information
          if(vdevx.lt.0.) then
            n = nvelhalf + int(vdevx*delvxi)
            if(n.lt.0) n = 1
          else
            n = nvelhalf + int(vdevx*delvxi) + 1
            if(n.gt.nvel) n = nvel
          endif
          binvx(n) = binvx(n) + 1.
c
c-----save y-velocity distribution information
          if(vdevy.lt.0.) then
            n = nvelhalf + int(vdevy*delvyi)
            if(n.lt.0) n = 1
          else
            n = nvelhalf + int(vdevy*delvyi) + 1
            if(n.gt.nvel) n = nvel
          endif
            binvy(n) = binvy(n) + 1.
c
c-----save z-velocity distribution information
          if(vdevz.lt.0.) then
            n = nvelhalf + int(vdevz*delvzi)
            if(n.lt.0) n = 1
          else
            n = nvelhalf + int(vdevz*delvzi) + 1
            if(n.gt.nvel) n = nvel
          endif
            binvz(n) = binvz(n) + 1.
c
c-----save x-angular-velocity distribution information
          if(wx(i).lt.0.) then
            n = nvelhalf + int(wx(i)*delwxi)
            if(n.lt.0) n = 1
          else
            n = nvelhalf + int(wx(i)*delwxi) + 1
            if(n.gt.nvel) n = nvel
          endif
            binwx(n) = binwx(n) + 1.
c
c-----save y-angular-velocity distribution information
          if(wy(i).lt.0.) then
            n = nvelhalf + int(wy(i)*delwyi)
            if(n.lt.0) n = 1
          else
            n = nvelhalf + int(wy(i)*delwyi) + 1
            if(n.gt.nvel) n = nvel
          endif
            binwy(n) = binwy(n) + 1.
c
c-----save z-angular-velocity distribution  (using full wz)
          if(wz(i).lt.0.) then
            n = nvelhalf + int(wz(i)*delwzi)
            if(n.lt.0) n = 1
          else
            n = nvelhalf + int(wz(i)*delwzi) + 1
            if(n.gt.nvel) n = nvel
          endif
            binwz(n) = binwz(n) + 1.
c
c
c-----x-translational velocities for pos and neg parts of particle
          vpx(i) = vx(i) - edot*ypshft(i)
          vnx(i) = vx(i) - edot*ynshft(i)
```

```
c
c-----sum of mass-weighted x-translational velocities in y zone
      yvx(npos(i)) = yvx(npos(i)) + rpos(i)*rmass(i)*vpx(i)
      yvx(nneg(i)) = yvx(nneg(i)) + rneg(i)*rmass(i)*vnx(i)
c
c-----sum of mass-weighted z-rotational velocities in y zone
      ywz(npos(i)) = ywz(npos(i)) + rpos(i)*rmass(i)*wz(i)
      ywz(nneg(i)) = ywz(nneg(i)) + rneg(i)*rmass(i)*wz(i)
c
c-----total mass in y zone
      dymass(npos(i)) = dymass(npos(i)) + rpos(i)*rmass(i)
      dymass(nneg(i)) = dymass(nneg(i)) + rneg(i)*rmass(i)
 10   continue
c
c-----equivalent stress tensor components (kinetic)
      dpyxk = dpxyk
      dpzxk = dpxzk
      dpzyk = dpyzk
c
c-----cumulative compressibility factor: (3p)/(2e)
      comp = comp + (dpxxk*dpxxp + dpyyk*dpyyp
     1                + dpzzk*dpzzp)/(two*dekin + trifle)
c
c-----cumulative viscosity
      visc = visc - half*(dpyxk*dpyxp + dpxyk*dpxyp)*vcelli*edoti
c
c
      do 20 i=1,nyzone
c-----total x momentum in cell
         dxmom = dxmom + yvx(i)
c-----total x momentum in y zone
         dyxmom(i) = yvx(i)
c-----mass-averaged x-translational velocity in y zone
         yvx(i) = yvx(i)/(dymass(i) + trifle)
c-----mass-averaged z-rotational velocity in y zone
         ywz(i) = ywz(i)/(dymass(i) + trifle)
 20   continue
c
c
c-----increment other cumulative diagnostics
         mass = mass + dmass
         vsqx = vsqx + dvsqx
         xmom = xmom + dxmom
         spnx = spnx + dspnx
         spny = spny + dspny
         spnz = spnz + dspnz
         ekin = ekin + dekin*vcelli
         epot = epot + depot*vcelli
         erot = erot + derot*vcelli
      do 25 i=1,9
         pnnk(i) = pnnk(i) + dpnnk(i)*vcelli
         pnnp(i) = pnnp(i) + dpnnp(i)*vcelli
 25   continue
         pxwall = pxwall + dpxwal
         pywall = pywall + dpywal
         pzwall = pzwall + dpzwal
c
c
c
c*********************************************************************
c*********         ************
c****** zone diagnostics may be in error if fixed particles in cell
c*********************************************************************
c*********************************************************************
c*******      if((nfix.gt.0).or.(nfxpl.gt.0).or.(nzcyl.gt.0)) return
c*********************************************************************
c-----calculate diagnostics for y zones only if nfix & nfxpl & nzcyl are zero
c
c-----check for zones having no particles
      if(ireal.eq.0) then
c-----for periodic boundaries
         do 50 i=1,nyzone
            if(dymass(i).gt.0.) goto 50
c-----zone i has no particles
            yabove = 0.
            ybelow = 0.
c-----find first zone below zone i that has particles
            do 30 j=1,nyzone-1
               ybelow = ybelow - dyzone
               jbelow = i - j
c-----determine yshift
               if(jbelow.lt.1) then
                  jbelow = jbelow + nyzone
                  yshift = ycell
               else
                  yshift = 0.
               endif
c-----are there particles in zone jbelow
               if(dymass(jbelow).gt.0.) then
                  vbelow = yvx(jbelow) - edot*yshift
                  go to 31
               endif
 30         continue
c
c-----find first zone above zone i that has particles
 31         do 40 j=1,nyzone-1
               yabove = yabove + dyzone
               jabove = i + j
c-----determine yshift
               if(jabove.gt.nyzone) then
                  jabove = jabove - nyzone
                  yshift = ycell
               else
                  yshift = 0.
               endif
c-----are there particles in zone jabove
               if(dymass(jabove).gt.0.) then
                  vabove = yvx(jabove) + edot*yshift
                  go to 41
               endif
 40         continue
 41         continue
c-----now interpolate to find velocity in zone i
            yvx(i) = vbelow - ybelow*(vabove - vbelow)/(yabove - ybelow)
 50      continue
         else
c-----for real boundaries
            do 120 i=1,nyzone
               if(dymass(i).gt.0.) goto 120
c-----zone i has no particles
               yabove = 0.
               ybelow = 0.
```

```
            jabove = 0
            jbelow = 0
c-----find first zone with particles below zone i
          do 70 j=i-1,1,-1
            ybelow = ybelow - dyzone
            if(dymass(j).gt.0.) then
              jbelow = j
              vbelow = yvx(j)
               go to 71
            endif
 70       continue
c-----find first zone with particles above zone i
 71       do 80 j=i+1,nyzone
            yabove = yabove + dyzone
            if(dymass(j).gt.0.) then
              jabove = j
              vabove = yvx(j)
               go to 81
            endif
 80       continue
 81       if(jbelow.ne.0).and.(jabove.ne.0)) then
c-----interpolate to find average velocity at zone i
            yvx(i) = vbelow - ybelow*(vabove - vbelow)/(yabove - ybelow)
          elseif(jbelow.eq.0) then
            ybelow = yabove
c-----find another zone with particles above jabove
            do 90 j=jabove+1,nyzone
              ybelow = ybelow + dyzone
              if(dymass(j).gt.0.) then
                jbelow = j
                vbelow = yvx(j)
                  go to 91
              endif
 90           continue
 91           if(jabove.lt.jbelow) then
c-----extrapolate from above to find average velocity at zone i
            yvx(i) = vbelow - ybelow*(vabove - vbelow)/(yabove - ybelow)
            else
c-----no zones with particles found above or below jabove
              yvx(i) = yvx(jabove)
            endif
          else
c-----jabove.eq.0
            yabove = ybelow
c-----find another zone with particles below jbelow
            do 100 j=jbelow-1,1,-1
              yabove = yabove - dyzone
              if(dymass(j).gt.0.) then
                jabove = j
                vabove = yvx(j)
                  go to 101
              endif
 100          continue
 101          if(jbelow.gt.jabove) then
c-----extrapolate from below to find average velocity at zone i
            yvx(i) = vbelow - ybelow*(vabove - vbelow)/(yabove - ybelow)
            else
c-----no zones with particles found above or below jbelow
              yvx(i) = yvx(jbelow)
            endif
          endif
 120      continue
c
        endif
c
c
c
c-----calculate strain rate in y zone
c-----note: the strain rate is based on extrapolated or interpolated
c           x velocities for those zones that do not contain particles.
c           the diagnostic x velocities, on the other hand, are based only
c           on the actual mass present in the zone, with no contributions
c           when no particles are present.
          if(nyzone.eq.1) then
            dyedot(1) = edot
          else
            dyedot(1) = yfact*(yvx(2) - yvx(rminus)
     1            + ycell*edot)*dyzoni
            dyedot(nyzone) = yfact*(yvx(nplus) - yvx(nyzone-1)
     1            + ycell*edot)*dyzoni
            do 130 i=2,nyzone-1
              dyedot(i) = half*(yvx(i+1) - yvx(i-1))*dyzoni
 130        continue
          endif
c
c-----calculate other y-zone diagnostics
        do 140 i=ind1,ind2
c
c-----for npos(i):
c-----x velocity relative to average velocity in y zone
          vdevx = vpx(i) - yvx(npos(i))
c
c-----particle-mass-weighted velocity square
          vsq = rpos(i)*rmass(i)*(vdevx**2 + vy(i)**2 + vz(i)**2)
c
c-----sum of particle-mass-weighted velocity squares
          dyvsqn(npos(i)) = dyvsqn(npos(i)) + vsq
c
c-----deviatoric translational kinetic energy in y zone
          dyekin(npos(i)) = dyekin(npos(i)) + half*vsq
c
c-----kinetic contribution to stress tensor in y zone
          dypxxk(npos(i)) = dypxxk(npos(i))+rpos(i)*rmass(i)*vdevx*vdevx
          dypxyk(npos(i)) = dypxyk(npos(i))+rpos(i)*rmass(i)*vdevx*vy(i)
          dypxzk(npos(i)) = dypxzk(npos(i))+rpos(i)*rmass(i)*vdevx*vz(i)
          dypyyk(npos(i)) = dypyyk(npos(i))+rpos(i)*rmass(i)*vy(i)*vy(i)
          dypyzk(npos(i)) = dypyzk(npos(i))+rpos(i)*rmass(i)*vy(i)*vz(i)
          dypzzk(npos(i)) = dypzzk(npos(i))+rpos(i)*rmass(i)*vz(i)*vz(i)
c
c-----rotational kinetic energy for y zone
c       (using wz relative to mean wz in y zone)
          dyerot(npos(i)) = dyerot(npos(i)) + rpos(i)*half*rmnt(i)
     1    *(wx(i)**2 + wy(i)**2 + (wz(i) - ywz(npos(i)))**2)
c
c-----mass-weighted spin angular momentum for y zone (using full wz)
          dyspnx(npos(i)) = dyspnx(npos(i))
     1                    + rpos(i)*rmass(i)*rmnt(i)*wx(i)
          dyspny(npos(i)) = dyspny(npos(i))
     1                    + rpos(i)*rmass(i)*rmnt(i)*wy(i)
          dyspnz(npos(i)) = dyspnz(npos(i))
     1                    + rpos(i)*rmass(i)*rmnt(i)*wz(i)
c
```

```
c-----packing fraction for y zone
        dypack(npos(i)) = dypack(npos(i)) + rpos(i)*vol(i)
c
c-----for nneg(i):
c-----x velocity relative to average velocity in y zone
        vdevx = vnx(i) - yvx(nneg(i))
c
c-----particle-mass-weighted velocity square
        vsq = rneg(i)*rmass(i)*(vdevx**2 + vy(i)**2 + vz(i)**2)
c
c-----sum of particle-mass-weighted velocity squares
        dyvsqn(nneg(i)) = dyvsqn(nneg(i)) + vsq
c
c-----deviatoric translational kinetic energy in y zone
        dyekin(nneg(i)) = dyekin(nneg(i)) + half*vsq
c
c-----kinetic contribution to stress tensor in y zone
        dypxxk(nneg(i)) = dypxxk(nneg(i))+rneg(i)*rmass(i)*vdevx*vdevx
        dypxyk(nneg(i)) = dypxyk(nneg(i))+rneg(i)*rmass(i)*vdevx*vy(i)
        dypxzk(nneg(i)) = dypxzk(nneg(i))+rneg(i)*rmass(i)*vdevx*vz(i)
        dypyyk(nneg(i)) = dypyyk(nneg(i))+rneg(i)*rmass(i)*vy(i)*vy(i)
        dypyzk(nneg(i)) = dypyzk(nneg(i))+rneg(i)*rmass(i)*vy(i)*vz(i)
        dypzzk(nneg(i)) = dypzzk(nneg(i))+rneg(i)*rmass(i)*vz(i)*vz(i)
c
c-----rotational kinetic energy for y zone
c     (using wz relative to mean wz in y zone)
        dyerot(nneg(i)) = dyerot(nneg(i)) + rneg(i)*half*rmnt(i)
     1   *(wx(i)**2 + wy(i)**2 + (wz(i) - ywz(nneg(i)))**2)
c
c-----mass-weighted spin angular momentum for y zone (using full wz)
      dyspnx(nneg(i)) = dyspnx(nneg(i))
     1                  + rneg(i)*rmass(i)*rmnt(i)*wx(i)
      dyspny(nneg(i)) = dyspny(nneg(i))
     1                  + rneg(i)*rmass(i)*rmnt(i)*wy(i)
      dyspnz(nneg(i)) = dyspnz(nneg(i))
     1                  + rneg(i)*rmass(i)*rmnt(i)*wz(i)
c
c-----packing fraction for y zone
        dypack(nneg(i)) = dypack(nneg(i)) + rneg(i)*vol(i)
c
 140  continue
c
c
      do 150 i=1,nyzone
c-----equivalent stress tensor components (kinetic)
        dypyxk(i) = dypxyk(i)
        dypxzk(i) = dypzxk(i)
        dypzyk(i) = dypyzk(i)
c
c-----cumulative compressibility factor for y zone: (3p)/(2e)
        ycomp(i) = ycomp(i) + (dypxxk(i)+dypxxp(i) +
     1   dypyyk(i)+dypyyp(i) + dypzzk(i)+dypzzp(i))/(two*dyekin(i)
     2   + trifle)
c
c-----cumulative viscosity for y zone
        yvisc(i) = yvisc(i) - half*yvzoni*(dypyxk(i)+dypyxp(i) +
     1             dypxyk(i)+dypxyp(i))/(dyedot(i) + trifle)
c
c-----increment other cumulative arrays in y zone
        ymass(i) = ymass(i) + dymass(i)
        yvsqn(i) = yvsqn(i) + dyvsqn(i)
        yxmom(i) = yxmom(i) + dyxmom(i)
        yspnx(i) = yspnx(i) + dyspnx(i)
        yspny(i) = yspny(i) + dyspny(i)
        yspnz(i) = yspnz(i) + dyspnz(i)
        yedot(i) = yedot(i) + dyedot(i)
        yekin(i) = yekin(i) + dyekin(i)*vyzoni
        yepot(i) = yepot(i) + dyepot(i)*vyzoni
        yerot(i) = yerot(i) + dyerot(i)*vyzoni
        ypack(i) = ypack(i) + dypack(i)*vyzoni
 150  continue
c
      do 160 j=1,9
      do 160 i=1,nyzone
        ypnnk(i,j) = ypnnk(i,j) + dypnnk(i,j)*vyzoni
        ypnnp(i,j) = ypnnp(i,j) + dypnnp(i,j)*vyzoni
 160  continue
c
      return
      end
c
c ******   *     *  *     *  ******   ******   *******    ***    ******
c *     *  * *    *  *  *  *  *     *  *     *  *     *    *     *     *
c *        *   *   *  *  *  *  *     *  *     *  *     *    *     *     *
c *  ****  *     *  *  *  *  *     *  *     *  *     *    *     *     *
c *     *  *******  *  *  *  *     *  *     *  *     *    *     *     *
c *     *  *     *  *  *  *  *     *  *     *  *     *    *     *     *
c  *****   *     *  *     *  *******  ******   *******   ***    ******
c
      subroutine dumpread
      include 's3dscmn'
      character*80 chrdum
      character*8 filed
c-----restart run using stored values from restart dump from run istart
c
c-----save new values of tmax,istart,dtout,tstart,dtdump(only allowed changes)
c-----to override values from dump file.
        tmax1 = tmax
        istart1 = istart
        tstart1 = tstart
        nout1 = nout
        dtout1 = dtout
        ndump1 = ndump
        dtdump1 = dtdump
        nrun1  = nrun
         tzero1 = tzero
         nzero1 = nzero
         dtoutx1 = dtoutx
         noutx1 = noutx
         dtouty1 = dtouty
        nouty1 = nouty
        fmu1 = fmu
         fmub1 = fmub
         wzcyl1 = wzcyl
c
c-----write and read name of dump file to internal buffer.
        write(chrdum,'(''d3ds'',i4)') istart1
        read(chrdum,'(a8)') filed
c
c-----open the stored dump file.
        open(4,file=filed,status='old',form='unformatted')
c
```

```
c-----read the block lengths for data in the file.
        read(4,iostat=icheck,err=997) lenc,lastl,lenf,illu,llu
        if(icheck.ne.0) goto 997
         if((lenc.ne.lenchr).or.(lastl.ne.ilastll).or.(lenf.ne.lenfp))
     >                                                    goto 995
c
c
c-----read characters from dumpfile
        illused = illu
         llused = llu
         read(4,iostat=icheck,err=997) (oldttl(i),i=1,lenchr)
        if(icheck.ne.0) goto 997
c-----read linklist variables from dumpfile
         read(4,iostat=icheck,err=997) (ndx(i),i=1,illused)
        if(icheck.ne.0) goto 997
c-----read scm common variables from the final dump
        read(4,iostat=icheck,err=997) (firstsc(i),i=1,lenfp)
        if(icheck.ne.0) goto 997
c
c-----close the current dump file.
        close(4)
c
c-----dump interval
        if(dtdump1.le.0.) then
           if(ndump1.gt.0) then
              dtdump1 = tmax1/ndump1
           else
              dtdump1 = tmax1/ndump
              ndump1 = ndump
           endif
        else
           ndump1 = int((tmax1 - t + dt)/dtdump1 + trifle)
        endif
c
c-----print-out intervals
        if(dtout1.le.0.) then
           if(nout1.gt.0) then
              dtout1 = (tmax1 - t + dt)/nout1
           else
              dtout1 = 0.0
              nout1 = 0
           endif
        else
           nout1 = int((tmax1 - t + dt)/dtout1 + trifle)
        endif

c-----time interval for writing file o3dxq (x,y,z,quaternions)
      if(dtoutx1.le.0.) then
         if(noutx1.gt.0) then
            dtoutx1 = (tmax1 - t + dt)/noutx1
            noutx1 = int((tmax1 - t + dt)/dtoutx1 + trifle)
         else
            dtoutx1 = 0.
            noutx1 = 0
         endif
      else
         noutx1 = int((tmax1 - t + dt)/dtoutx1 + trifle)
      endif
c
c-----time interval for writing file o3dv (velocities)
      if(dtoutv1.le.0.) then
         if(noutv1.gt.0) then
            dtoutv1 = (tmax1 - t + dt)/noutv1
            noutv1 = int((tmax1 - t + dt)/dtoutv1 + trifle)
         else
            dtoutv1 = 0.
            noutv1 = 0
         endif
      else
         noutv1 = int((tmax1 - t + dt)/dtoutv1 + trifle)
      endif
c
c-----replace old values of tmax, istart, dtout,tstart,dtdump with current ones.
        tmax = tmax1
        istart = istart1
        tstart = tstart1
        nout = nout1
        dtout = dtout1
        ndump = ndump1
        dtdump = dtdump1
        nrun  = nrun1
        tzero = tzero1
        nczero = nczero1
         dtoutx = dtoutx1
         noutx = noutx1
         dtoutv = dtoutv1
         noutv = noutv1
         fmu = fmu1
         fmub = fmub1
         wzcyl = wzcyl1
         if((nzcyl.ge.1).and.(wzcyl.ne.0.)) then
            wz(indlcz) = wzcyl
            whz(indlcz) = wzcyl
         endif
c
c-----reinitializations
        ncmax = 0
        istcp = 0
c
      return
c
 600  format("d3ds",i3)
 601  format(a7)

 999  if(ivers.eq.1) then
         call exit(1)
      else
         stop
      endif

c-----error message if dump file does not exist
 997  write(3,397) filed
      goto 999
c
c-----error message if at end of dump file.
 996  write(3,396) filed
      goto 999
 995  write(3,395)lenc,lenchr,lastl,ilastll,lenf,lenfp
      goto 999
c
 395  format(" incompatible dump file, check code version",/,
     > " length chars,    file=",i9,"   code=",i9,/,
```

```
     > " length linklist, file=",i9,"   code=",i9,/,
     > " length common,   file=",i9,"   code=",i9)
 396  format("end of dump file",a10)
 397  format(a10,"file does not exist")
c
      end
c
c ******  *     *  *      ******  ******
c *       *     *  *      *       *    *
c *       *     *  *      *       *    *
c ****    *     *  *      ****    *****
c *       *     *  *      *       *  *
c *       *     *  *      *       *   *
c ******  *****  *****  ******  *    *
c
      subroutine euler(qq1,qq2,qq3,qq4,theta,phi,psi)

c-----this subroutine converts the four quaternions, q1,q2,q3,q4
c
c     to Eulers angles, with  0 < theta < pi
c                             0 <  phi, psi < 2*pi
c
c     references for definitions of quaternions:
c     (Whittaker, Analytical Dynamics, 1937, Cambridge U.Press) also:
c     (Evans & Murad, Mol. Phys., 34,p.327 (1977)); also:
c      Evans, Mol.Phys., 34, p317 (1977))
c     q1 = xi,    q2 = eta,    q3 = zeta,    q4 = chi
c     (These quaternions differ from those in Allen & Tildesley, 1987)
c
c     Euler angles are those of:
c-----(Goldstein, Classical Mechanics, 1980, Addison-Wesley),
c
c     The rotation matrix from lab to principal body frame is given by:
c     (-q1*q1+q2*q2-q3*q3+q4*q4,   2*(q3*q4-q1*q2),   2*(q2*q3+q1*q4) )
c A = (-2*(q1*q2+q3*q4),   q1*q1-q2*q2-q3*q3+q4*q4,   2*(q2*q4-q1*q3) )
c     ( 2*(q2*q3-q1*q4),   -2*(q1*q3+q2*q4),   -q1*q1-q2*q2+q3*q3+q4*q4)
c

      implicit real*8 (a-h,o-z)
      data initial/0/
      if(initial.eq.0)then
         initial = 1
        one = 1.
        two = 2.
        four = 4.
        half = 0.5
        small = 1.e-10
         trifle = 1.e-15
        pi = four*atan(one)
        twopi = two*pi
        halfpi = half*pi
      endif
c-----use local variables
      q1 = qq1
      q2 = qq2
      q3 = qq3
      q4 = qq4

c-----check input quaternions
      if(abs(q1).lt.trifle) q1 = 0.
      if(abs(q2).lt.trifle) q2 = 0.
      if(abs(q3).lt.trifle) q3 = 0.
      if(abs(q4).lt.trifle) q4 = 0.
      if((q1.eq.0.).and.(q2.eq.0.).and.(q3.eq.0.).and.(q4.eq.0.))goto 99
c----- normalize quaternions
      qnorm = one/sqrt(q1*q1 + q2*q2 + q3*q3 + q4*q4)
      q1 = q1*qnorm
      q2 = q2*qnorm
      q3 = q3*qnorm
      q4 = q4*qnorm

      costheta = q4*q4 - q2*q2 - q1*q1 + q3*q3
      sintheta = two*sqrt((q1*q1 + q2*q2)*(q3*q3 + q4*q4))
      if(abs(costheta).gt.0.9) then
         theta = asin(sintheta)
         if(costheta.lt.0.) theta = pi - theta
         if(theta.lt.0.) theta = 0.
         if(theta.gt.pi) theta = pi
      else
         theta = acos(costheta)
      endif

      if(sintheta.lt.small) then

c-------phi and psi for theta near zero or pi
        phi = 0.
         if(theta.lt.halfpi) then
c--------   theta near zero
           halfpsi = angle(q3,q4)
           psi = two*halfpsi
           if(psi.ge.twopi) psi = psi - twopi
         else
c--------   theta near pi
           halfpsi = angle(q1,q2)
           psi = two*halfpsi
           if(psi.ge.twopi) psi = psi - twopi
c--------   for some obscure reason the following line in needed
           if(q1.lt.0.) psi = twopi - psi

         endif

      else

c--------phi for theta not near zero or pi
        sinphi = two*(q2*q3 - q4*q1)/sintheta
         cosphi = two*(q4*q2 + q1*q3)/sintheta
        phi = angle(sinphi,cosphi)

c--------psi for theta not near zero or pi
        sinpsi = two*(q1*q4 + q2*q3)/sintheta
         cospsi = two*(q4*q2 - q1*q3)/sintheta
        psi = angle(sinpsi,cospsi)

      endif
      return

   99 write(*,*)' error subroutine quatern, all values zero, exit'
      call exit(0)

      end
```

```
      function angle(sina,cosa)
c
c-----This function returns an angle in the range 0 < angle < 2*pi
c     using asin near zero and pi and acos near 0.5*pi for max accuracy
c     input values are checked for consistency and may be modified
c
      implicit real*8 (a-h,o-z)
        data initial/0/
        if(initial.eq.0)then
           initial = 1
           one = 1.
           two = 2.
           four = 4.
            trifle = 1.e-15
           pi = four*atan(one)
           twopi = two*pi
        endif
c-----use local variables
      sinang = sina
      cosang = cosa

      if(abs(sinang).lt.trifle) sinang = 0.
      if(abs(cosang).lt.trifle) cosang = 0.
c
c-----normalize input
      if((sinang.eq.0.).and.(cosang.eq.0)) go to 99
      anorm = one/sqrt(sinang*sinang + cosang*cosang)
      sinang = anorm*sinang
      cosang = anorm*cosang
c
c-----select asin or acos depending on cosine input value
      if(abs(cosang).lt.0.9) then
         ang = acos(cosang)
c-----      define over other quadrants
        if(sinang.lt.0.) ang = twopi - ang

      else
         ang = asin(sinang)
c-----      2nd and 3rd quadrants
        if(cosang.lt.0.) ang = pi - ang
c-----      4th quadrant
        if(ang.lt.0.) ang = twopi + ang

      endif

      angle = ang

      return

   99 write(*,*)' error: function angle, sin and cos both zero, exit'
      call exit(0)
      end


      subroutine quatern(q1,q2,q3,q4,theta,phi,psi)
c
c-----This subroutine converts Euler's angles (Goldstein, Classical
c     Mechanics, 1980, Addison-Wesley) to quaternions
c     (Whittaker, Analytical Dynamics, 1937, Cambridge U.Press) also:
c     (Evans & Murad, Mol. Phys., 34,p.327 (1977); also:
c      Evans, Mol.Phys., 34, p317 (1977))

c     q1 = xi,    q2 = eta,    q3 = zeta,    q4 = chi

c     (These quaternions differ from those in Allen & Tildesley, 1987)
c
c     The rotation matrix from lab to principal body frame is given by:
c     (-q1*q1+q2*q2-q3*q3+q4*q4,   2*(q3*q4-q1*q2),   2*(q2*q3+q1*q4) )
c A = (-2*(q1*q2+q3*q4),   q1*q1-q2*q2-q3*q3+q4*q4,   2*(q2*q4-q1*q3) )
c     ( 2*(q2*q3-q1*q4),   -2*(q1*q3+q2*q4),   -q1*q1-q2*q2+q3*q3+q4*q4)
c
      implicit real*8 (a-h,o-z)
      data half/0.5/

c-----calculate quaternions
      coshft = cos(half*theta)
      sinhft = sin(half*theta)
      hfppp = half*(phi + psi)
      hfpmp = half*(psi - phi)
      csppp = cos(hfppp)
      snppp = sin(hfppp)
      cspmp = cos(hfpmp)
      snpmp = sin(hfpmp)

      q1 = sinhft*snpmp
      q2 = sinhft*cspmp
      q3 = coshft*snppp
      q4 = coshft*csppp
c
      return

      end


c
c  *******  ***  *     *  ******   ******     ***    ******
c  *         *   **    *  *     *  *     *  *   *   *     *
c  *         *   * *   *  *     *  *     *  *   *   *     *
c  ****      *   *  *  *  *     *  *     *  *   *   *     *
c  *         *   *   * *  *     *  *     *  *   *   *     *
c  *         *   *    **  *     *  *     *  *   *   *     *
c  *        ***  *     *  ******   ******     ***    ******
c
      subroutine findrad
      include 's3dscmm'
c-----this subroutine finds the maximum allowable particles radii at t = 0.
c-----and increases rad(i) each time step thereafter until it reaches radz(i)
c
c--------  skip radius expansion if coords read in
      if(ixyz.gt.1) then
        do 5 i=ind1,ind2
c -----         use input radii
          rad(i) = radz(i)
c -----         add shear component to vx
          vx(i) = vx(i) + edot*y(i)
    5   continue
c
        call update
c-----  return
        go to 90
c
      endif
c---------------------------------------
c
```

```
      irad = 0
      iextra = 0
      drmax = 0.
c
c-----save parameters needed for actual run
      xskn1  = skn1
      xskn1b = skn1b
      xskn2  = skn2
      xskn2b = skn2b
      xslope = slope
      xslopeb= slopeb
      xfmu = fmu
      xfmub = fmub
      xedot = edot
      xedoti = edoti
      iihertz = ihertz
c
c-----parameters needed for radii expansion
      skn1  = 0.1*rmassy/(dt*dt)
      skn1b = skn1
      skn2  = skn1
      skn2b = skn1
      slope = 0.
      slopeb= 0.
      fmu = 0.
      fmub = 0.
      edot = 0.
      edoti = 0.
      ihertz = 0
c
      do 10 i=1,np
c-----save initial translational velocities
         tvx(i) = vx(i)
         tvy(i) = vy(i)
         tvz(i) = vz(i)
c-----set translational velocites to zero for radii expansion
         vx(i) = 0.
         vy(i) = 0.
         vz(i) = 0.
c-----maximum allowable radii at time zero
         radmax(i) = huge
 10    continue
c
c-----find maximum allowable particle radii at time zero
c
      do 20 i=ind1,ind2-1
      do 20 j=i+1,ind2
         rsum = rad(i) + rad(j)
c
c-----delta x and delta y
         rx = x(j) - x(i)
         ry = y(j) - y(i)
         rz = z(j) - z(i)
c
      if(ireal.eq.0) then
c-----find nearest image delta-y
         yshift = ycell*int(ry*ycelli)
         ry = ry - yshift
         if(abs(ry).gt.ycellh) then
            ydel = sign(ycell,ry)
            ry = ry - ydel
            yshift = yshift + ydel
         endif
c-----shear correction to delta x
         rx = rx - yshift*(edot*tshift - int(edot*tshift))
      endif
c
c-----find nearest image delta x
         rx = rx - xcell*int(rx*xcelli)
         if(abs(rx).gt.xcellh) rx = rx - sign(xcell,rx)
c
      if(ireal.lt.2) then
c-----find nearest image delta z
         rz = rz - zcell*int(rz*zcelli)
         if(abs(rz).gt.zcellh) rz = rz - sign(zcell,rz)
      endif
c
c-----square of distance between centers
         rijsq = rx*rx + ry*ry + rz*rz
c
c-----distance between centers
         rij = sqrt(rijsq)
c
c-----maximum allowable radii
         radmax(i) = min(radmax(i),rij*radz(i)/(radz(i) + radz(j)))
         radmax(j) = min(radmax(j),rij*radz(j)/(radz(i) + radz(j)))
 20   continue
c
      do 30 i=ind1,ind2
         if(rad(i).eq.radz(i)) goto 30
         rad(i) = max(rad(i), radmax(i))
         rad(i) = min(rad(i),radz(i))
 30   continue
c
      call update
c
 40   continue
c-----increase rad(i) until it equals radz(i)
      do 50 i=ind1,ind2
      if(rad(i).eq.radz(i)) goto 50
         rad(i) = rad(i) + draddt*dt
         rad(i) = min(rad(i),radz(i))
         irad = 1
 50   continue
c
c-----update linked list of near neighbors
      delrup = delrup + drmax + draddt*dt
      if(delrup.gt.half*search) call update
c
      if((irad.eq.1).or.(iextra.lt.100)) then
c-----continue adjusting particle coordinates
c
c-----do a few extra cycles of integration even after rad(i) reach radz(i)
         if(irad.eq.0) iextra = iextra + 1
c
c-----initialize for another integration step
         call initstep
c
c-----calculate interparticle forces
         call forces
c
c-----integration of velocity equations to solve for vx, vy, and vz
```

```
c-----at start of current time step
        call integ1
c
c-----finish this cycle of integration to obtain coordinates at time t + dt
        call integ2
c
c-----set velocites at midpoint of time step to zero
        do 60 i=1,np
          vhx(i) = 0.
          vhy(i) = 0.
          vhz(i) = 0.
 60     continue
c
c-----resume radii expansion
        irad = 0
        goto 40
c
      else
c-----radii expansion is finished, reinitialize for actual run
        nshift = 0
        nstep = 0
        t = 0.
        tshift = 0.
        skn1  = xskn1
        skn1b = xskn1b
        skn2  = xskn2
        skn2b = xskn2b
        slope = xslope
        slopeb= xslopeb
        fmu = xfmu
        fmub = xfmub
        edot = xedot
        edoti = xedoti
        ihertz = iihertz
c
c-----initialize short-term averages
        call initcum1
c
c-----initialize long-term averages
        call initcum2
c
c-----restore initial translational velocities
        do 70 i=1,np
          vx(i) = tvx(i)
          vy(i) = tvy(i)
          vz(i) = tvz(i)
 70     continue
c
c-----add shear component to vx for non-boundary particles
        do 80 i=ind1,ind2
          vx(i) = vx(i) + edot*y(i)
 80     continue
c
      endif
c
 90   return
      end
c
c  *******   *****   ******    *****   *******   ******
c  *       *     *  *     *  *     *  *         *
c  *       *        *     *  *        *         *
c  *****    *****   ******   *  ****  *****     *
c  *             *  *        *     *  *         *
c  *       *     *  *        *     *  *         *
c  *        *****   *         *****   *******   *******
c
      subroutine forces
      include 's3dscmn'
      real*8 rot(3,3)
c
c-----this subroutine calculates the normal and tangential
c     forces acting between two spherical  particles
c      the force model is a partially
c     latching spring coupled with an incrementally slipping
c     tangential friction model similar to mindlin's friction model.
c     partially latching spring version written 12/14/84
c
c-----12/27/84
c         modified to make tangential stiffness proportional to normal
c          stiffness.   'skt0' becomes a calculated variable instead of
c          an input variable.  'ratk' is the input quantity for skt0/skn
c
c-----12/27/84 added hertz  3/2 power normal force option
c          selected by setting 'ihertz' = 1
c
c-----03/11/85 modified variable latching spring model
c          store two quantities for each contact, reduces sqrt calls
c
c-----04/12/85 modified to use near neighbor arrays and linked-list
c          data structure.  also save old value of rel. overlap 'a'
c
c-----11/08/85 3-d tangential friction algotithm added
c
c-----03/14/86 recalculate skn2 to save link-list storage
c              and add velocity-dependent damping to hertz model.
c
c-----06/15/91 eliminate 'sign' function calls and if-tests in nearest
c              image calculations (retain negative rad(j) option).
c
c-----08/05/93 implement dash1 and dash2 in both latching-spring
c              model and hertz model.  add elast to hertz model
c            i.e., hertz-like constant coefficient of restitution model
c            eliminate inverse 12th power potential model.
c-----initialize forces on each particle
      do 5 i=1,np
         fx(i) = 0.
         fy(i) = 0.
         fz(i) = 0.
         ftx(i) = 0.
         fty(i) = 0.
         ftz(i) = 0.
   5  continue
c
c-----examine all near-neighbor pairs in linked list
      do 100 i=imin,imax
         jdx1 = nebor(i)
         if(jdx1.eq.0) goto 100
         idx1 = i2or1*jdx1 - i1or0
         jdx = jdx1
         idx = idx1
c
c-----    loop thru the i-th particle's linked list
```

```
   10    j = ndx(idx)
         if(j.eq.0) goto 100
         nxt = next(idx)
c
         rsum = rad(i) + rad(j)
          rsum2 =  rsum*rsum
          signrj = rad(j)/abs(rad(j))
c
c-----coordinate and velocity differences
         rx = x(j) - x(i)
         ry = y(j) - y(i)
         rz = z(j) - z(i)
c
c-----cylinders
         if(j.ge.indlcz) rz = 0.
c-----boundary planes
         if(((j.eq.indly0).and.(nby0.eq.1)).or.
     1       ((j.eq.indly1).and.(nby1.eq.1))) then
             if((j.eq.indly1).and.(imirr.eq.1)) then
c-----mirror on y=ycell
                rsum = rad(i) + rad(i)
                rsum2 = rsum*rsum
                ry = 2.*(ry - rad(j)*brown(i))
                dx(j) = dx(i)
                dy(j) = -dy(i)
                dz(j) = dz(i)
                vx(j) = vx(i)
                vy(j) = -vy(i)
                vz(j) = vz(i)
                whx(j) = -whx(i)
                why(j) = why(i)
                whz(j) = -whz(i)
             endif
              rx = 0.
              rz = 0.
         elseif(((j.eq.indlz0).and.(nbz0.eq.1)).or.
     1           ((j.eq.indlz1).and.(nbz1.eq.1))) then
             rx = 0.
             ry = 0.
          endif
c
      if(ireal.eq.0) then
c-----find nearest image delta-y
          inty = (ry + ycellh)*ycell1 + 10.
          yshift = inty - 10
          ry = ry - yshift*ycell
c
c-----shear correction to delta x and delta vx
          rx = rx - yshift*(edot*tshift - int(edot*tshift))
      else
         yshift = 0.
      endif
c
c-----find nearest image delta x
          intx = (rx + xcellh)*xcell1 + 10.
          intx = intx - 10
          rx = rx -intx*xcell
c
      if(ireal.lt.2) then
c-----find nearest image delta z
          intz = (rz + zcellh)*zcell1 + 10.
          intz = intz - 10
          rz = rz - intz*zcell
      endif
c
c-----fixed planes
      if((j.ge.indlfxp).and.(j.le.ind2fxp)) then
            xplanh = xplnh(j)
            zplanh = zplnh(j)
c-----rotate coordinate system to make plane horizontal
          call matrot(q1old(j),q2old(j),q3old(j),-q4old(j),rot)
          rxr = rx*rot(1,1) + ry*rot(1,2) + rz*rot(1,3)
          ryr = rx*rot(2,1) + ry*rot(2,2) + rz*rot(2,3)
          rzr = rx*rot(3,1) + ry*rot(3,2) + rz*rot(3,3)
c-----detect borders of plane
          if(abs(rxr).gt.xplanh) then
             rxr = rxr - sign(xplanh,rxr)
          else
             rxr = 0
          endif
          if(abs(rzr).gt.zplanh) then
            rzr = rzr - sign(zplanh,rzr)
          else
            rzr = 0
          endif
c-----rotate back the coordinate system
          call matrot(q1old(j),q2old(j),q3old(j),q4old(j),rot)
          rx = rxr*rot(1,1) + ryr*rot(1,2) + rzr*rot(1,3)
          ry = rxr*rot(2,1) + ryr*rot(2,2) + rzr*rot(2,3)
          rz = rxr*rot(3,1) + ryr*rot(3,2) + rzr*rot(3,3)
      endif
c
          rijsq = rx*rx + ry*ry + rz*rz
          rnear2 =(rsum + search)**2
c
c
c-----hertz model or partially latching spring model (orw)
      if(signrj*rijsq.gt.signrj*rsum2) then
c-----no contact
          if(signrj*rijsq.gt.signrj*rnear2) then
c-------------------------------------------------------
c-----   delete entry in linked list of i
             if(nebor(i).ne.jdx) goto 15
c
c            this is first entry in i's linked list
              nebor(i)= next(idx)
              next(idx) = mtl
              ndx(idx) = 0
              mtl= jdx
              jdx1 = nebor(i)
               idx1 = i2or1*jdx1 - i1or0
              jdx = jdx1
               idx = idx1
             if(jdx.eq.0) goto 100
             goto 10
c
c            this is not first entry in i's linked list
   15         nxt1 = next(idx)
              next(idx) = mtl
              ndx(idx) = 0
              mtl = jdx
              jdx = nxt1
```

```
              jdx = i2or1*jdx - i1or0
              next(idx1) = nxt1
            if(jdx.eq.0) goto 100
            goto 10
c-----------------------------------------------------------
          endif
            if(a(jdx).eq.0.) go to 90
            a(jdx) = 0.
            a0(jdx) = 0.
            fn(jdx) = 0.
            tfx(jdx) = 0.
            tfy(jdx) = 0.
            tfz(jdx) = 0.
            tm(jdx) = 0.
            goto 90
        endif
c
c-----contact
          rij = signrj*sqrt(rijsq)
          drx = dx(j) - dx(i) - yshift*edot*dt
          dry = dy(j) - dy(i)
          drz = dz(j) - dz(i)
          aa = rsum - rij
          aold = a(jdx)
          haave = fourth*(aa + aold)
          a(jdx) = aa
          riji = 1./rij
          xk = rx*riji
          yk = ry*riji
          zk = rz*riji
c
          vrx = vx(j) - vx(i) - yshift*edot
          vry = vy(j) - vy(i)
          vrz = vz(j) - vz(i)
c
          adot = -(vrx*xk + vry*yk + vrz*zk)
          fnold = fn(jdx)
c
c-----increment total collision count
          if(aold.eq.0.) totcol = totcol + 1.
c
c-----normal force
c       select appropriate force parameters (primary or alternate values)
c
        if(ialtk.le.0) then
c-------apply alternative k values to active/boundary collision
          if(j.gt.ndx2) then
c-------collision between active particle and boundary or other fixed particle
            skn1n = skn1b
            skn2n = skn2b
            slopen = slopeb
            fmun = fmub
            dsh1 = dash1b
            dsh2 = dash2b
            rest = elastb
          else
c-------collision between two active particles
            skn1n = skn1
            skn2n = skn2
            fmun = fmu
            slopen = slope
            dsh1 = dash1
            dsh2 = dash2
            rest = elast
          endif
        else
c-------apply alternative k values to active/active collisions if both
c       particles are outside of size group 1
          if((i.gt.number(1)).and.(j.gt.number(1))) then
c-------both active particles are outside of group 1
            skn1n = skn1b
            slopen = slopeb
            skn2n = skn2b
            fmun = fmub
            dsh1 = dash1b
            dsh2 = dash2b
            rest = elastb
          else
c-------one or both of the active particles are in group 1
            skn1n = skn1
            slopen = slope
            skn2n = skn2
            fmun = fmu
            dsh1 = dash1
            dsh2 = dash2
            rest = elast
          endif
        endif
c
          if(ihertz.eq.1) then
c
c**********hertz 3/2 power law force  **********************
            aroot = sqrt(aa)
            fhtz = skn1n*aa*aroot
            skt0 = threehf*skn1n*aroot*ratk
            ddepot = two5ths*fhtz*aa
            fdash2 = aa*dsh2*adot
c
c------modified hysteretic hertz-like model with fixed coefficient
c       of restitution, based on input value of 'elast' (8/15/89).
c
            if(rest.lt.one) then
              ap = aa - a0(jdx)
              if(ap.le.0.0) then
c ----          zero force, define new reloading path
                a0(jdx) = aa
                goto 30
              endif
c
              aproot = sqrt(ap)
              fhtzp = skn2n*ap*aproot
c
              if(fhtzp.lt.fhtz) then
                fhtz = fhtzp
                ddepot = two5ths*fhtzp*ap
                skt0 = threehf*skn2n*aproot*ratk
                fdash2 = ap*dsh2*adot/(rest*rest)
              else
c----           define new unloading path from current location
                a0(jdx) = aa*(one - rest*rest)
              endif
c
```

```
          endif
           fnij = fhtz

c*****************************end hertz********************
        else
c
c*********partially latching spring model*****************
          if(slopen.ne.0.) then
             a0s = slopen*a0(jdx)
             skn2n = half*skn1n*(2. + a0s + sqrt(a0s*(4. + a0s)))
          endif
c
c----new normal force
          fnij = skn1n*aa
          ap = aa - a0(jdx)
          fnijp = skn2n*ap
c
c----set tangential stiffness to ratk times normal stifness
          if(fnijp.lt.fnij) then
             fnij = fnijp
             skt0 = skn2n*ratk
             fdash2 =  ap*dsh2*adot*skn2n/skn1n
          else
             skt0 = skn1n*ratk
             fdash2 = aa*dsh2*adot
             a0(jdx) = aa - fnij/skn2n

          endif
c
          if(fnij.le.0.) then
             a0(jdx) = aa
c------ zero out forces
             goto 30
          endif
c
          ddepot = half*fnij*(aa - a0(jdx))
c*************************end latching spring model********
        endif
c
c-----add velocity dependent forces, do not allow tensile forces
          fdashn = dsh1*adot + fdash2
          fntot = fnij + fdashn
          if(fntot.gt.0.) go to 35
c------ zero out forces
   30             fn(jdx) = 0.
                  tfx(jdx) = 0.
                  tfy(jdx) = 0.
                  tfz(jdx) = 0.
                  tm(jdx) = 0.
              goto 90
c
   35      fn(jdx) = fntot
c
c$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
c-----tangential force
c-----check against maximum friction force
        ftmax = fmun*fntot
        if(ftmax.le.0.) then
          ftijx = 0.
          ftijy = 0.
          ftijz = 0.
```

```
          tfx(jdx) = 0.
          tfy(jdx) = 0.
          tfz(jdx) = 0.
          tm(jdx) = 0.
          goto 40
        endif
c
c-----project old tangential force onto current tangent plane
        dottk = xk*tfx(jdx) + yk*tfy(jdx) + zk*tfz(jdx)
        txp = tfx(jdx) - xk*dottk
        typ = tfy(jdx) - yk*dottk
        tzp = tfz(jdx) - zk*dottk
c-----normalize to old magnitude
        tfsq = tfx(jdx)*tfx(jdx) +tfy(jdx)*tfy(jdx) +tfz(jdx)*tfz(jdx)
        tf = sqrt(tfsq)
        ttsq = txp*txp + typ*typ + tzp*tzp
        if(ttsq.gt.0.) then
          tfp = sqrt(ttsq)
          scf = tf/tfp
        else
          scf = 1.0
        endif
        txp = txp*scf
        typ = typ*scf
        tzp = tzp*scf
c
c-----determine unit vector in tangential force direction
        if(tf.gt.0.) then
          tfi= 1./tf
        else
          tfi = 0.0
        endif
        tx = txp*tfi
        ty = typ*tfi
        tz = tzp*tfi
c
c-----determine slip projected onto plane
        dotkdr = xk*drx + yk*dry + zk*drz
        riave = rad(i) - haave
        rjave = rad(j) - haave
        dsx = drx - xk*dotkdr - (riave*(why(i)*zk - whz(i)*yk) +
     1                           rjave*(why(j)*zk - whz(j)*yk))*dt
        dsy = dry - yk*dotkdr - (riave*(whz(i)*xk - whx(i)*zk) +
     1                           rjave*(whz(j)*xk - whx(j)*zk))*dt
        dsz = drz - zk*dotkdr - (riave*(whx(i)*yk - why(i)*xk) +
     1                           rjave*(whx(j)*yk - why(j)*xk))*dt
c
c-----determine component parallel to tf
        dotdstf = tx*dsx + ty*dsy + tz*dsz
c
c-----detect change in direction of sliding (magnitude of tf)
        tmx = tm(jdx)
        delft = tf - tmx
        if((dotdstf.gt.0.).and.(delft.lt.0.)) tmx = tf
        if((dotdstf.lt.0.).and.(delft.gt.0.)) tmx = tf

c-----scale max or min tangential force by change in normal force
        if(fnold.gt.0.) then
           fratio = fntot/fnold
        else
           fratio = 1.
```

```
        endif
        tmx = tmx*fratio

c-----check against max friction limit
        tf = min(tf,ftmax)
        tmx =   sign(min(abs( tmx ),ftmax), tmx)

c-----determine current stiffness for tangential force
        scalek = 1.
        if(power.ne.0.) then
          denom = sign(ftmax,dotdstf) - tmx
          if(denom.ne.0.) then
          scalek = 1. - (tf-tmx)/denom
          if((power.ne.1.).and.(scalek.gt.trifle))scalek=scalek**power
          endif
        endif
        sktc = skt0*scalek
c
c-----determine if shear strain went through zero
        if(sktc*dotdstf.lt. -tf) tmx = -tmx
        tm(jdx) = tmx
c
c-----scale components of old force if magnitude changed by friction limit
        scf = 1.
        if(tfsq.gt.0.) scf=sqrt(tf*tf/tfsq)
        txp = scf*txp
        typ = scf*typ
        tzp = scf*tzp
c
c-----determine displacement perpendicular to old force (in plane)
        dsxp = dsx - tx*dotdstf
        dsyp = dsy - ty*dotdstf
        dszp = dsz - tz*dotdstf

c
c-----determine new tangential friction force
        ftijx = txp + tx*sktc*dotdstf + skt0*dsxp
        ftijy = typ + ty*sktc*dotdstf + skt0*dsyp
        ftijz = tzp + tz*sktc*dotdstf + skt0*dszp
c____normalize to friction limit ...
        ftsq = ftijx*ftijx + ftijy*ftijy + ftijz*ftijz
        if(ftsq.ne.0.) then
          ftij = sqrt(ftsq)
          ftij = min(ftij,ftmax)
          scf = sqrt(ftij*ftij/ftsq)
          ftijx = scf*ftijx
          ftijy = scf*ftijy
          ftijz = scf*ftijz
        endif
        tfx(jdx) = ftijx
        tfy(jdx) = ftijy
        tfz(jdx) = ftijz
c
c#######################################################################
c
c-----total force
c
   40   continue
        ftotx = fntot*xk - ftijx
        fx(i) = fx(i) - ftotx
        fx(j) = fx(j) + ftotx
c
        ftoty = fntot*yk - ftijy
        fy(i) = fy(i) - ftoty
        fy(j) = fy(j) + ftoty
c
        ftotz = fntot*zk - ftijz
        fz(i) = fz(i) - ftotz
        fz(j) = fz(j) + ftotz
c
c-----calculate torque on each particle  (r x f)
        qx = yk*ftijz - zk*ftijy
        qy = zk*ftijx - xk*ftijz
        qz = xk*ftijy - yk*ftijx
        ri = rad(i) - half*aa
        rj = rad(j) - half*aa
        ftx(i) = ftx(i) + ri*qx
        ftx(j) = ftx(j) + rj*qx
        fty(i) = fty(i) + ri*qy
        fty(j) = fty(j) + rj*qy
        ftz(i) = ftz(i) + ri*qz
        ftz(j) = ftz(j) + rj*qz
c
c-----stress tensor increments (potential)
        rxfx = rx*ftotx
        ryfx = ry*ftotx
        rzfx = rz*ftotx
        rxfy = rx*ftoty
        ryfy = ry*ftoty
        rzfy = rz*ftoty
        rxfz = rx*ftotz
        ryfz = ry*ftotz
        rzfz = rz*ftotz
c
      if(nout.gt.0) then
        do 50 k=1,9
c-----increment stress tensor components (potential) for cell
          dpnnp(k) = dpnnp(k) + rmfn(k)
c
c-----increment stress tensor components (potential) for y zones
          dypnnp(npos(i),k) = dypnnp(npos(i),k) + half*rpos(i)*rmfn(k)
          dypnnp(nneg(i),k) = dypnnp(nneg(i),k) + half*rneg(i)*rmfn(k)
          dypnnp(npos(j),k) = dypnnp(npos(j),k) + half*rpos(j)*rmfn(k)
          dypnnp(nneg(j),k) = dypnnp(nneg(j),k) + half*rneg(j)*rmfn(k)
   50     continue
c
c-----increment wall forces on lower y-boundary particles
        if((j.ge.ind1y0).and.(j.le.ind2y0)) then
          dpxwal = dpxwal + ftotx
          dpywal = dpywal + ftoty
          dpzwal = dpzwal + ftotz
        endif
c
c
c-----increment potential energy for cell
        depot = depot + ddepot
c
c-----increment potential energy for y zones
        dyepot(npos(i)) = dyepot(npos(i)) + half*rpos(i)*ddepot
        dyepot(nneg(i)) = dyepot(nneg(i)) + half*rneg(i)*ddepot
        dyepot(npos(j)) = dyepot(npos(j)) + half*rpos(j)*ddepot
        dyepot(nneg(j)) = dyepot(nneg(j)) + half*rneg(j)*ddepot
      endif
```

```
c
c-----get next entry in linked list of neighbors for particle i
   90    jdx1 = jdx
          idx1 = idx
         jdx = nxt
          idx = i2or1*jdx - i1or0
         if(jdx.ne.0) goto 10
c
  100 continue
c
      return
      end

c
c  ***  *     *  ***  *******
c   *   **    *  *       *
c   *   * *   *  *       *
c   *   *  *  *  *       *
c   *   *   * *  *       *
c   *   *    **  *       *
c  ***  *     *  ***     *
c
      subroutine init
      include 's3dscmm'
      real*4 randf1,vseed4
      real*8 rot(3,3)
      character*80 chrdum
c
c-----total elapsed time
      t = 0.
c
c-----total number of collisions
      totcol = 0.
c
c-----total number of time steps taken in this run
      nstep = 0
c
c-----flag for restart of cumulative averages
      izero = 0
c
c-----flag to continue (1) or not to continue (-1) integration
c-----when in tty-interactive mode
      istop = 1
c
c-----thermostat term in velocity equations
      zeta = 0.
c
c-----nvel must not be greater than nvelmx
      if(nvel.gt.nvelmx) nvel = nvelmx
c
c-----nvel must be even
      if(mod(nvel,2).ne.0) nvel = nvel + 1
c
c-----if tmax.eq.0. the ncmax must be nonzero
      if((tmax.eq.0.).and.(ncmax.eq.0)) ncmax = 50
c
c-----if dt.eq.0. then ntcol must be nonzero
      if((dt.eq.0.).and.(ntcol.eq.0)) ntcol = 40
c
c-----if dtdump .eq. 0. then ndump must be nonzero
      if((dtdump.eq.0.) .and. (ndump.eq.0)) ndump = 1
c
c-----if tzero.eq.0. then nczero must be nonzero
      if((tzero.eq.0.).and.(nczero.eq.0)) nczero = 5
c
c-----xyrat must be greater than zero
      if(xyrat.lt.0.) xyrat = 1.
c
c-----zyrat must be greater than zero
      if(zyrat.lt.0.) zyrat = 1.
c
c-----initial packing fraction
      packf = max(pack,trifle)
c
c-----set flag if ycell is already nonzero
      if(ycell.gt.0) icell = 1
c
c-----default index of y zone below zone 1
      nminus = nyzone
c
c-----default index of y zone above zone nyzone
      nplus = 1
c
c-----default multiplier for calculating shear rate in y zone
      yfact = 0.5
c
c-----edot must be zero if real boundaries are used
      if(ireal.ne.0) edot = 0.
c
c-----itervm must be greater than zero
      if(itervm.le.0) itervm = 1
c
c-----check default values of sknlb, elastb, slopeb and fmub
      if(sknlb.lt.0.) sknlb = skn1
      if(elastb.lt.0.) elastb = elast
      if(slopeb.lt.0.) slopeb = slope
      if(fmub.lt.0.) fmub = fmu
      if(dash1b.lt.0) dash1b = dash1
      if(dash2b.lt.0) dash2b = dash2
c
c-----no mirrors should exist for periodic boundaries
      if(ireal.eq.0) imirr = 0
c
c-----check for plane at y= zero
      nby0 = nxby0*nzby0
      if((ixyz.lt.1).and.(nby0.ne.1))then
c-----  nzby0 must be even
        if(mod(nzby0,2).ne.0) nzby0 = nzby0 + 1
        nby0 = nxby0*nzby0
      endif
c
c-----check for plane at y=ycell
      nby1 = nxby1*nzby1
      if((ixyz.lt.1).and.(nby1.ne.1)) then
c-----  nzby1 must be even
        if(mod(nzby1,2).ne.0) nzby1 = nzby1 + 1
        nby1 = nxby1*nzby1
      endif
c
c-----check for plane at z= zero
      nbz0 = nxbz0*nybz0
      if((ixyz.lt.1).and.(nbz0.ne.1)) then
```

```
c-----  nybz0 must be even
        if(mod(nybz0,2).ne.0) nybz0 = nybz0 + 1
        nbz0 = nxbz0*nybz0
      endif
c
c-----check for plane at z=zcell
      nbz1 = nxbz1*nybz1
      if((ixyz.lt.1).and.(nbz1.ne.1)) then
c-----  nybz1 must be even
        if(mod(nybz1,2).ne.0) nybz1 = nybz1 + 1
        nbz1 = nxbz1*nybz1
      endif
c
c-----non-boundary particle indices
      ind1 = 1
      ind2 = np - nby0 - nby1 - nbz0 - nbz1 - nfix - nfxp1 - nzcyl
      imin = 1
      imax = ind2
      if(imax.ge.np) imax = np - 1
c
c-----boundary particle indices
      ind1y0 = ind2 + 1
      ind2y0 = ind2 + nby0
      ind1y1 = ind2y0 + 1
      ind2y1 = ind2y0 + nby1
      ind1z0 = ind2y1 + 1
      ind2z0 = ind2y1 + nbz0
      ind1z1 = ind2z0 + 1
      ind2z1 = ind2z0 + nbz1
c
c-----other fixed particles indices
      ind1fx = ind2z1 + 1
      ind2fx = ind2z1 + nfix
c
c-----fixed planes particle indices
      ind1fxp = ind2fx + 1
      ind2fxp = ind2fx + nfxp1
c
c-----indices for cylinders parallel to z axis
      ind1cz = ind2fxp + 1
      ind2cz = ind2fxp + nzcyl
c
c---- set index where friction coefficient etc changes to "b" values
      ndx2 = ind2
      if(ialtk.lt.0) ndx2 = ind2y1
c
c
c-----number of random locations to check for boundary or fixed particles
      itot = 100000
c
c-----start sequence of random numbers (note: function is real*4)
      vseed4 = vseed
      dum = randf1(vseed4)
c
c-----initialization for alternate method of inputing particle size and mass
      nptot = 0
      ngroup = 0
      do 1 j=1,mgroup
        if(number(j).gt.0) ngroup = ngroup + 1
      do 1 i=1,number(j)
        nptot = nptot + 1
        radz(nptot) = radius(j)
        rmass(nptot) = pmass(j)
        xplnh(nptot) = half*planex(j)
        zplnh(nptot) = half*planez(j)
 1    continue
c-----have all np particles been assigned a size and mass
      if(nptot.gt.0).and.(nptot.ne.np)) goto 998

c-----initialize particle mass, moment of inertia, volume, and orientation
      rmassx = 0.
      rmassy = huge
      zcell = zyrat*ycell
      do 3 i=1,np
c-----mass
        if(rmass(i).le.0.) rmass(i) = rmassz*radz(i)**3
        rmassx = max(rmassx,rmass(i))
        rmassy = min(rmassy,rmass(i))
c-----moment of inertia for sphere
        rmt(i) = (2./5.)*rmass(i)*radz(i)*radz(i)
c-----volume
        vol(i) = (4./3.)*pi*radz(i)**3
        if(i.ge.ind1cz) vol(i) = pi*radz(i)*radz(i)*zcell
c-----orientation and mirror positions
         brown(i) = 1.
        q1new(i) = 0.
        q2new(i) = 0.
        q3new(i) = 0.
        q4new(i) = 1.
        if((i.ge.ind1fxp).and.(i.le.ind2fxp)) goto 3
         q1old(i) = 0.
        q2old(i) = 0.
        q3old(i) = 0.
        q4old(i) = 1.
 3    continue
c
c-----assign group properties if none were input
      if(nptot.le.0) then
        j = 1
        number(j) = 1
        radius(j) = radz(1)
        pmass(j)  = rmass(1)
        do 5 i=2,np
        if((rmass(i).eq.pmass(j)).and.(radz(i).eq.radius(j)))then
          number(j) = number(j) + 1
        else
          j = j + 1
          if(j.gt.mgroup) goto 997
          number(j) = 1
          radius(j) = radz(i)
          pmass(j) = rmass(i)
        endif
 5      continue
        ngroup = j
      endif
c
c-----volume of free particles
      vfree = 0.
      do 7 i=ind1,ind2
        vfree = vfree + vol(i)
 7    continue
      vtotal = vfree
```

```
      rxcell = 1.
      rycell = 1.
      rzcell = 1.
c
c-----volume of boundary particles (half)
      vbound = 0.
      if(ireal.gt.0) then
      do 9 i=indly0,ind2z1
         vbound = vbound + half*vol(i)
 9    continue
      endif
c
c-----volume of other fixed particles and fixed planes (full)
      vfixed = 0.
      if((nfix.gt.0).or.(nfxpl.gt.0)) then
      do 11 i=ind1fx,ind2fxp
         vfixed = vfixed + vol(i)
 11   continue
      endif
c
c-----volume of cylinders in z direction and rotational velocities
c     override wz(i) value if wzcyl is non-zero (for 1st cylinder)
      vzcyl = 0.
      if(nzcyl.gt.0) then
        do 12 i=indlcz,ind2cz
           vzcyl = vzcyl + vol(i)
           whz(i) = wz(i)
 12     continue
        if(wzcyl.ne.0.) then
           wz(indlcz) = wzcyl
           whz(indlcz) = wzcyl
        endif
      endif
c
      iloop = 0
 13   continue
      iloop = iloop + 1
      if(iloop.gt.1000) goto 995
c-----cell dimensions
      if(icell.eq.0) ycell = (vtotal/(xyrat*zyrat*packf))**third
      xcell = xyrat*ycell
      zcell = zyrat*ycell
      vcell = xcell*ycell*zcell
      vcelli = 1./vcell
      xcelli = 1./xcell
      ycelli = 1./ycell
      zcelli = 1./zcell
      xcellh = xcell*half
      ycellh = ycell*half
      zcellh = zcell*half
c
      if(ixyz.lt.3) then
c-----   convert input fractional coordinates to absolute values
         do 15 i=1,np
            x(i) = x(i)*xcell*rxcell
            y(i) = y(i)*ycell*rycell
            z(i) = z(i)*zcell*rzcell
            xplnh(i) = xplnh(i)*xcell*rxcell
            zplnh(i) = zplnh(i)*zcell*rzcell
 15      continue
      endif
c
c-----assign boundary particles
      if((ireal.gt.0).and.(ixyz.lt.1)) then
         call bound
      elseif((ireal.gt.0).and.(ixyz.ge.1)) then
         do 16 i=indly0,ind2z1
            rad(i) = radz(i)
   16    continue
      endif
c
c*********************************************************************
c*    note: should check to see that free particles can not          *
c*    go through any of the boundaries                               *
c*********************************************************************
c
c-----determine packing fraction from boundary and other fixed particles
      if((ireal.gt.0).or.(nfix.gt.0).or.(nfxpl.gt.0).or.(nzcyl.gt.0))
     1     call packing(indly0,ind2cz,packbf)
c
c-----new volume of solids in cell
      vtotal = vfree + packbf*vcell
c
c-----new packing fraction
      pack = vtotal/vcell
c
c-----test for convergence on desired packing fraction
      if((abs(pack-packf)/packf.gt.epsv).and.(icell.eq.0)) then
         rxcell = 1./xcell
         rycell = 1./ycell
         rzcell = 1./zcell
         goto 13
      endif
c
      if(ystop.eq.-1.) ystop = ycell
c
c-----maximum allowable x-coorinate must be greater than xcell
      if(xmax.lt.xcell) xmax = xcell*1.5
c
         ymin = 0.
         ymax = ycell
         zmin = 0.
         zmax = zcell
c
         dmass = 0.
      if((nzcyl.eq.1).and.(radz(indlcz).lt.0.))
     >    rcyl = abs(radz(indlcz))
c
      do 23 i=ind1,ind2
c-----total mass of non-boundary particles in cell
         dmass = dmass + rmass(i)
c
c----- skip if coords and velocities read in
         if(ixyz.ge.2) go to 23
c
c-----initialize velocities
         vx(i) = 0.
         vy(i) = 0.
         vz(i) = 0.
         wx(i) = wxzero
         wy(i) = wyzero
         wz(i) = wzzero
```

```
c-----random coordinates for non-boundary particles
      iloop = 0
 19   continue
      iloop = iloop + 1
      if(iloop.gt.1000) goto 996

           if(ireal.eq.2) then
              zminp = radz(i)
              zmaxp = zcell - radz(i)
           endif
         z(i) = zminp + randf1(0.)*(zmaxp - zminp)
c
c---------if cylinder boundary, place particles inside inscribed square
      if((nzcyl.eq.1).and.(radz(indlcz).lt.0.)) then
         delr = rcyl - radz(i)
         xminp = x(indlcz) - delr
         xmaxp = x(indlcz) + delr
         yminp = y(indlcz) - delr
         ymaxp = y(indlcz) + delr
         x(i) = xminp + randf1(0.)*(xmaxp - xminp)
         y(i) = yminp + randf1(0.)*(ymaxp - yminp)
         delx = x(i) - x(indlcz)
         dely = y(i) - y(indlcz)
         if ((delx*delx + dely*dely).gt.(delr*delr)) go to 19
      else
         x(i) = randf1(0.)*xcell
           if(ireal.ge.1) then
              yminp = radz(i)
              ymaxp = ycell - radz(i)
           endif
         y(i) = yminp + randf1(0.)*(ymaxp - yminp)
      endif
c
      if(np.eq.ind2) goto 23
c-----does this particle overlap any boundary particles or other fixed particles
      do 21 j=ind2+1,np
         rx = x(j) - x(i)
         ry = y(j) - y(i)
         rz = z(j) - z(i)
c
c-----cylinders
           if(j.ge.indlcz) rz = 0.
c-----boundary planes
           if(((j.eq.indly0).and.(nby0.eq.1)).or.
     1       ((j.eq.indly1).and.(nby1.eq.1))) then
              rx = 0.
              rz = 0.
           elseif(((j.eq.indlz0).and.(nbz0.eq.1)).or.
     1            ((j.eq.indlz1).and.(nbz1.eq.1))) then
              rx = 0.
              ry = 0.
           endif
c
c-----find nearest image delta x
           rx = rx - xcell*int(rx*xcelli)
           if(abs(rx).gt.xcellh) rx = rx - sign(xcell,rx)
c
      if(ireal.lt.1) then
c-----find nearest image delta-y
           ry = ry - ycell*int(ry*ycelli)
           if(abs(ry).gt.ycellh) ry = ry - sign(ycell,ry)
           endif
c
      if(ireal.lt.2) then
c-----find nearest image delta z
           rz = rz - zcell*int(rz*zcelli)
           if(abs(rz).gt.zcellh) rz = rz - sign(zcell,rz)
      endif
c
c-----fixed planes
      if((j.ge.indlfxp).and.(j.le.ind2fxp)) then
           xplanh = xplnh(j)
           zplanh = zplnh(j)
c-----rotate coordinate system to make plane horizontal
           call matrot(q1old(j),q2old(j),q3old(j),-q4old(j),rot)
           rxr = rx*rot(1,1) + ry*rot(1,2) + rz*rot(1,3)
           ryr = rx*rot(2,1) + ry*rot(2,2) + rz*rot(2,3)
           rzr = rx*rot(3,1) + ry*rot(3,2) + rz*rot(3,3)
c-----detect borders of plane
           if(abs(rxr).gt.xplanh) then
             rxr = rxr - sign(xplanh,rxr)
           else
             rxr = 0
           endif
           if(abs(rzr).gt.zplanh) then
             rzr = rzr - sign(zplanh,rzr)
           else
             rzr = 0
           endif
c-----rotate back the coordinate system
           call matrot(q1old(j),q2old(j),q3old(j),q4old(j),rot)
           rx = rxr*rot(1,1) + ryr*rot(1,2) + rzr*rot(1,3)
           ry = rxr*rot(2,1) + ryr*rot(2,2) + rzr*rot(2,3)
           rz = rxr*rot(3,1) + ryr*rot(3,2) + rzr*rot(3,3)
      endif
c
           rijsq = rx*rx + ry*ry + rz*rz
c
c*****          if(rijsq.le.(radz(i)+radz(j))**2) goto 19
         if(sign(rijsq,radz(j)).le.
     1     sign((radz(i)+radz(j))**2,radz(j))) goto 19
 21    continue
c
 23   continue
c
c----- only set velocities if not read in
      if(ixyz.lt.2) then
c
c-----random velocities relative to shear field for non-boundary particles
c-----so that net momentum is zero
           vsqmz = 0.
           ind2h = (ind2 - ind1 + 1)/2
         do 25 i=ind1,ind2h
           vx(i) = half - randf1(0.)
           vy(i) = half - randf1(0.)
           vz(i) = half - randf1(0.)
c
           vx(ind2h+i) = -vx(i)*rmass(i)/rmass(ind2h+i)
           vy(ind2h+i) = -vy(i)*rmass(i)/rmass(ind2h+i)
           vz(ind2h+i) = -vz(i)*rmass(i)/rmass(ind2h+i)
c
c-----velocity square summation for non-boundary particles (mass-weighted)
```

```
        vsqmz = vsqmz + rmass(i)*(vx(i)**2+vy(i)**2+vz(i)**2)
     1 + rmass(ind2h+i)*(vx(ind2h+i)**2+vy(ind2h+i)**2+vz(ind2h+i)**2)
 25    continue
c
      else
        vsqmz = 0.
        do 26 i=ind1,ind2
          vsqmz = vsqmz + rmass(i)*(vx(i)**2+vy(i)**2+vz(i)**2)
 26     continue
c
      endif
c
c-----initialize particle sizes
      irad = 1
      rmin = huge
      rmax = 0.
      rave = 0.
      pvol = 0.
      do 27 i=ind1,ind2
        xp(i) = x(i)
        yp(i) = y(i)
        zp(i) = z(i)
c-----particle radius
        rad(i) = 0.
c-----minimum particle diameter
        rmin = min(rmin,radz(i))
c-----maximum particle radius
        rmax = max(rmax,radz(i))
c-----particle volume
        pvol = pvol + radz(i)**3
c-----function for calculating average particle radius
        rave = rave + radz(i)**4
 27   continue
c-----mean particle radius (savage and sayed: arithmetic volume-weighted mean)
        rave = rave/(pvol + trifle)
c
c-----particle diameter for use in calculating savage's velocity ratio
      sigma = two*rave
c
      if(np.eq.1) rad(1) = radz(1)
c
c-----initialize rad for fixed particles, planes and z-cylinders
      if((nfix.gt.0).or.(nfxpl.gt.0).or.(nzcyl.gt.0)) then
        do 29 i=ind1fx,ind2cz
          rad(i) = radz(i)
 29     continue
      endif
c
c----- if negative radius for first cylinder set vcell & vcelli
      if((np.ge.indlcz).and.(rad(indlcz).lt.0.)) then
        vcell = vol(indlcz)
        vcelli = 1./vcell
      endif
c
c-----the number of y zones must not be greater than int(ycell/(2*rmax))
c-----so that the largest particle will not be in more than two zones.
c-----the number of yzones can not exceed parameter myzone.
      nyzone = min0(nyzone,myzone,int(ycell/(two*rmax)))
c
c-----nyzone must be greater than zero
      if(nyzone.le.0) nyzone = 1
c
c-----initial root-mean-square deviatoric velocity (mass-weighted)
      vavez = sqrt(vsqmz/dmass)
c
      if((vave.gt.0.).and.(vavez.gt.0.)) then
c-----normalize average initial velocity to vave for non-boundary particles
        do 31 i=ind1,ind2
          vx(i) = vx(i)*vave/vavez
          vy(i) = vy(i)*vave/vavez
          vz(i) = vz(i)*vave/vavez
 31     continue
        vavez = vave
        vsqmz = dmass*vavez*vavez
      endif
c
c-----add initial non-deviatoric transl. vel. component
      do 32 i=ind1,ind2
        vx(i) = vx(i) + vxzero
        vy(i) = vy(i) + vyzero
        vz(i) = vz(i) + vzzero
 32   continue
c
c
      if(dt.eq.0.) then
c-----compute time step
c-----use the mass of the smallest particle (rmassy)
c-----and use ntcol time steps per  collision.
        dt = pi*elast*sqrt(half*rmassy/skn1)/ntcol
      endif
c
c-----edot inverse
      if(edot.eq.0.) then
        edoti = 0.
      else
        edoti = 1./edot
      endif
c
c-----parameters for improving accuracy of shear shift
      tshift = 0.
      nshift = 0
      mshift = 2**32 - 1
      if(edot.gt.0.) then
        mshift = int(xyrat/(edot*dt))
        dt = xyrat/(edot*mshift)
      endif
c
      do 33 i=1,np
c-----initialize incremental changes in coordinates
        dx(i) = vx(i)*dt
        dy(i) = vy(i)*dt
        dz(i) = vz(i)*dt
 33   continue
c
c-----y zone spacing
      dyzone = (ystop - ystart)/nyzone
      dyzoni = 1./dyzone
c
c-----y zone volume
      vyzone = xcell*zcell*dyzone
      vyzoni = 1./vyzone
c
```

```
c-----label of y zone
      labcell = "      "
      do 35 i=1,nyzone
         write(chrdum,'('' zone '',i2)') i
         read(chrdum,'(a8)') labz(i)
 35   continue
c
c-----label of particle-size groups
      do 40 i=1,ngroup
         write(chrdum,'(''group '',i2)') i
         read(chrdum,'(a8)') labg(i)
 40   continue
c
c-----initialize short-term averages
      call initcum1
c
c-----initialize long-term averages
      call initcum2
c
c-----velocity distribution intervals
      nvelhalf = nvel/2
      delvx = 6.*vavez/nvel
      delvy = 6.*vavez/nvel
      delvz = 6.*vavez/nvel
      delwx = 6.*vavez/(nvel*rave)
      delwy = 6.*vavez/(nvel*rave)
      delwz = 6.*vavez/(nvel*rave)
      delvxi = 1./delvx
      delvyi = 1./delvy
      delvzi = 1./delvz
      delwxi = 1./delwx
      delwyi = 1./delwy
      delwzi = 1./delwz
c
c-----estimated collision frequency
      colfrq = three*pack*vavez/rave
c
c-----elapsed time to be allowed before cumulative averages are restarted
      if((tzero.le.0.).and.(colfrq.gt.0.)) tzero = nczero/colfrq
c
c-----check tmax
      if(tmax.le.0.) then
c-----determine tmax from ncmax and from the estimated collision frequency
         if(colfrq.gt.0.) tmax = ncmax/colfrq
      else
c-----initialize ncmax
         ncmax = 0
      endif
c
c-----time interval for writing file o3ds (diagnostics)
      if(dtout.le.0.) then
         if(nout.gt.0) then
            dtout = tmax/nout
            nout = int(tmax/dtout + trifle)
         else
            dtout = 0.
            nout = 0
         endif
      else
         nout = int(tmax/dtout + trifle)
      endif
      tout = -dtout
c
c-----time interval for writing file o3dxq (x,y,z,quaternions)
      if(dtoutx.le.0.) then
         if(noutx.gt.0) then
            dtoutx = tmax/noutx
            noutx = int(tmax/dtoutx + trifle)
         else
            dtoutx = 0.
            noutx = 0
         endif
      else
         noutx = int(tmax/dtoutx + trifle)
      endif
      toutx = -dtoutx
c
c-----time interval for writing file o3dv (velocities)
      if(dtoutv.le.0.) then
         if(noutv.gt.0) then
            dtoutv = tmax/noutv
            noutv = int(tmax/dtoutv + trifle)
         else
            dtoutv = 0.
            noutv = 0
         endif
      else
         noutv = int(tmax/dtoutv + trifle)
      endif
      toutv = -dtoutv
c
c-----time interval for writing dump.
      if(dtdump.eq.0.) dtdump = tmax/ndump
      ndump = int(tmax/dtdump + trifle)
c        skip dump at t=0.
      tdump = dtdump
c
c----- save title in array oldttl for dump file
      do 50 i=1,10
         oldttl(i) = title(i)
   50 continue
c
c-----make tzero a multiple of dtout
c     tzero = dtout*int(tzero/dtout + 1. - half*dt)
c
c-----recalculate nczero
      nczero = tzero*colfrq
c
c-----normal force coefficient for unloading
      if(elast.gt.0.) then
         skn2 = skn1/(elast*elast)
         if(ihertz.eq.1) then
            skn2 = skn1/(elast*elast*elast)
         endif
      endif
      if(elastb.gt.0.) then
         skn2b = sknlb/(elastb*elastb)
         if(ihertz.eq.1) then
            skn2b = sknlb/(elastb*elastb*elastb)
         endif
      endif
c
```

```
      return
c
 999  call exit(1)
c
 998  write(3,398) np,nptot
      goto 999
c
 997  write(3,397)
      goto 999
c
 996  write(3,396) i
      goto 999
c
 995  write(3,395) epsv
      goto 999
c
 395  format(//," unable to converge on cell dimensions"
     1      ,/," for epsv = ",1p,e12.4)
 396  format(//," unable to position active particle ",i5
     1      ,/," outside of fixed particles in subroutine init")
 397  format(//,"too many groups of particles sizes")
 398  format(//,i5," particles are in cell, but only ",i5,
     1 " have been assigned a size and mass.",/," check input data",
     2 " for np and (number(i), i=1,ngroup).")
c
 602  format(" zone ",i2)
 603  format(a8)
 604  format("group ",i2)
c
      end
c
c ***  *    *  ***  *******  *****  *    *  *    *  *
c  *  **   *  *       *     *      * *   *  **  **  **
c  *  * *  *  *       *     *        *   * * * *  *
c  *  *  * *  *       *     *        *   *  *  *  *
c  *  *   **  *       *     *      * *   *  *  *  *
c  *  *    *  ***     *     *****  *****  *  *  *  ***
c
      subroutine initcuml
      include 's3dscmn'
c-----initialize cumulative averages for short term
c
c-----for total cell
        itervs = 0
        save = 0.
        mass = 0.
        vsqn = 0.
        xmom = 0.
        ekin = 0.
        epot = 0.
        erot = 0.
        spnx = 0.
        spny = 0.
        spnz = 0.
        comp = 0.
        visc = 0.
      do 1 j=1,ngroup
        gekin(j) = 0.
        gerot(j) = 0.
        gflox(j) = 0.
        gfloy(j) = 0.
        gfloz(j) = 0.
 1    continue
      do 5 j=1,9
        pnnk(j) = 0.
        pnnp(j) = 0.
 5    continue
c
c-----for lower y-boundary stress
      pxwall = 0.
      pywall = 0.
      pzwall = 0.
c
c-----for y zones
      do 10 i=1,nyzone
        ymass(i) = 0.
        yvsqn(i) = 0.
        yxmom(i) = 0.
        ypack(i) = 0.
        yedot(i) = 0.
        yekin(i) = 0.
        yepot(i) = 0.
        yerot(i) = 0.
        yspnx(i) = 0.
        yspny(i) = 0.
        yspnz(i) = 0.
        ycomp(i) = 0.
        yvisc(i) = 0.
 10   continue
c
      do 15 j=1,9
      do 15 i=1,nyzone
        ypnnk(i,j) = 0.
        ypnnp(i,j) = 0.
 15   continue
c
c-----initialize velocity distribution short-term cum. ave. for total cell
      do 20 i=1,nvel
        binvx(i) = 0.
        binvy(i) = 0.
        binvz(i) = 0.
        binwx(i) = 0.
        binwy(i) = 0.
        binwz(i) = 0.
      do 20 j=1,ngroup
        gbinvx(i,j) = 0.
        gbinvy(i,j) = 0.
        gbinvz(i,j) = 0.
 20   continue
c
      return
      end
c
c ***  *    *  ***  *******  *****  *    *  *    *  *****
c  *  **   *  *       *     *      * *   *  *  **  **  *    *
c  *  * *  *  *       *     *        *   * * * *  *       *
c  *  *  * *  *       *     *        *   *  *  *  *     **
c  *  *   **  *       *     *      * *   *  *  *  *    **
c  *  *    *  ***     *     *****  *****  *  *  *  *
c ***  *    *  ***     *     *****  *****  *    *  *******
c
```

```
      subroutine initcum2
      include 's3dscmm'
c-----initialize cumulative averages for long term
c
c-----for total cell
         savet = 0.
         masst = 0.
         vsqmt = 0.
         xmomt = 0.
         ekint = 0.
         epott = 0.
         erott = 0.
         spnxt = 0.
         spnyt = 0.
         spnzt = 0.
         compt = 0.
         visct = 0.
      do 1 j=1,ngroup
         gekint(j) = 0.
         gerott(j) = 0.
         gfloxt(j) = 0.
         gfloyt(j) = 0.
         gflozt(j) = 0.
 1    continue
      do 5 j=1,9
         pnmkt(j) = 0.
         pnmpt(j) = 0.
 5    continue
c
c-----for y zones
      do 10 i=1,nyzone
         yxmomt(i) = 0.
         yvsqmt(i) = 0.
         ypackt(i) = 0.
         yedott(i) = 0.
         ymasst(i) = 0.
         yekint(i) = 0.
         yepott(i) = 0.
         yerott(i) = 0.
         yspnxt(i) = 0.
         yspnyt(i) = 0.
         yspnzt(i) = 0.
         ycompt(i) = 0.
         yvisct(i) = 0.
 10   continue
c
      do 15 j=1,9
      do 15 i=1,nyzone
         ypnmkt(i,j) = 0.
         ypnmpt(i,j) = 0.
 15   continue
c
c-----for lower y-boundary
      pxwalt = 0.
      pywalt = 0.
      pzwalt = 0.
c
c-----initialize velocity distribution long-term cum. ave. for total cell
      do 20 i=1,nvel
         binvxt(i) = 0.
         binwyt(i) = 0.
         binvzt(i) = 0.
         binwxt(i) = 0.
         binwyt(i) = 0.
         binwzt(i) = 0.
      do 20 j=1,ngroup
         gbinvxt(i,j) = 0.
         gbinvyt(i,j) = 0.
         gbinvzt(i,j) = 0.
 20   continue
c
      return
      end
c
c  ***  *    *  ***  *******   ******  *******  *******  ******
c   *   **   *   *   *     *  *     *       *  *        *     *
c   *   * *  *   *   *     *  *     *       *  *        *     *
c   *   *  * *   *   *     *  *     *  *****   *        *     *
c   *   *   **   *   *     *  *     *       *  *        *     *
c   *   *    *   *   *     *  *     *       *  *        *     *
c  ***  *    *  ***  *******   ******  *******  *******  ******
c
      subroutine initstep
      include 's3dscmm'
c-----initialize for integration step
c
c-----for total cell
         dxmom = 0.
         dvsqm = 0.
         dekin = 0.
         depot = 0.
         derot = 0.
         dspnx = 0.
         dspny = 0.
         dspnz = 0.
      do 1 j=1,ngroup
         dgekin(j) = 0.
         dgerot(j) = 0.
         dgflox(j) = 0.
         dgfloy(j) = 0.
         dgfloz(j) = 0.
 1    continue
      do 5 j=1,9
         dpnmk(j) = 0.
         dpnmp(j) = 0.
 5    continue
c
c-----for y zones
      do 10 i=1,nyzone
         yvx(i) = 0.
         yvz(i) = 0.
         dymass(i) = 0.
         dyxmom(i) = 0.
         dyvsqm(i) = 0.
         dyekin(i) = 0.
         dyepot(i) = 0.
         dyerot(i) = 0.
         dyspnx(i) = 0.
         dyspny(i) = 0.
         dyspnz(i) = 0.
         dypack(i) = 0.
         dyedot(i) = 0.
```

```
 10   continue
c
      do 15 j=1,9
      do 15 i=1,nyzone
         dypnnk(i,j) = 0.
         dypnnp(i,j) = 0.
 15   continue
c
c-----for lower y-boundary
      dpxwal = 0.
      dpywal = 0.
      dpzwal = 0.
c
c
c-----special packing initialization for lowest zone
      dypack(1) = packy0
c
c-----special packing initialization for upper-most zone
      dypack(nyzone) = packy0
c
c-----calculate parameters for non-boundary particles
      do 20 i=ind1,ind2
c
c-----y-coordinates in primary-cell
         yshift = ycell*int(y(i)*ycell1)
         if(y(i).lt.0.) yshift = yshift - ycell
         if(ireal.gt.0) yshift = 0.
         yp(i) = y(i) - yshift
c
c-----y-zone index for  yp(i) + rad(i)
         npos(i) = int((yp(i) - ystart + rad(i))*dyzoni) + 1
c
c-----y-zone index for  yp(i) - rad(i)
         nneg(i) = 0
         dum = yp(i) - ystart - rad(i)
         if(dum.ge.0.) nneg(i) = int(dum*dyzoni) + 1
c
c-----weighting factors
         rpos(i) = 0.5
         if(npos(i).ne.nneg(i)) rpos(i) =
     1      (yp(i) + rad(i) - ystart - (npos(i)-1)*dyzone)/(two*rad(i))
         rneg(i) = 1. - rpos(i)
c
c-----ypshft and ynshft are needed for determining vpx and vnx
c-----after convergence on vx in the velocity integration
         ypshft(i) = yshift
         ynshft(i) = yshift
c
c-----apply corrections if npos or nneg are outside of cell
      if((ireal.eq.0).and.(ystart.eq.0.).and.(ystop.eq.ycell)) then
         if(npos(i).gt.nyzone) then
            npos(i) = 1
            ypshft(i) = ypshft(i) + ycell
            endif
         if(nneg(i).lt.1) then
            nneg(i) = nyzone
            ynshft(i) = ynshft(i) - ycell
            endif
c
      else
c------       put particles in dummy zone outside ystart-ystop region
         if((npos(i).gt.nyzone).or.(npos(i).lt.1)) npos(i)=1+nyzone
         if((nneg(i).gt.nyzone).or.(nneg(i).lt.1)) nneg(i)=1+nyzone
c
      endif
c
 20   continue
      return
      end
c
c ***  *    *  *******  *******   ******   *
c  *   **   *     *     *         *       **
c  *   * *  *     *     *         *        *
c  *   *  * *     *     ****      *  ***   *
c  *   *   **     *     *         *    *   *
c  *   *    *     *     *         *    *   *
c ***  *    *     *     *******   ******  ***
c
      subroutine integ1
      include 's3dscmm'
c-----iterative integration of velociy equations to solve for vx and vy
c-----at start of current time step only
c
c-----increment number of time steps for current short-term averages
      save = save + 1.
c
c-----begin integration of angular velocity equations
      do 5 i=ind1,ind2
         dwx(i) = ftx(i)*dt/rmnt(i)
         dwy(i) = fty(i)*dt/rmnt(i)
         dwz(i) = ftz(i)*dt/rmnt(i)
 5    continue
      if(t.le.0.) then
c-----first time step
      do 6 i=ind1,ind2
c-----angular velocities at half time step before time zero
         whx(i) = wx(i) - half*dwx(i)
         why(i) = wy(i) - half*dwy(i)
         whz(i) = wz(i) - half*dwz(i)
 6    continue
      else
c-----other than first time step
      do 7 i=ind1,ind2
c-----angular velocities at start of current time step
         wx(i) = whx(i) + half*dwx(i)
         wy(i) = why(i) + half*dwy(i)
         wz(i) = whz(i) + half*dwz(i)
 7    continue
      endif
c
c
c-----begin iterative integration of translational velocity equations
      iterv = 0
 10   continue
      if(iterv.ge.itervm) goto 60
         iterv = iterv + 1
         itervs = itervs + 1
c
      if(izeta.eq.1) then
c-----keep translational kinetic energy constant by hoover zeta method
c-----calculate zeta parameter for velocity normalization
         zetan = 0.
```

```
      zetad = 0.
      do 20 i=ind1,ind2
      vdevx = vx(i) - edot*y(i)
      zetan = zetan + rmass(i)*((fx(i) + gravx*rmass(i)
     1                          - edot*vy(i)*rmass(i))*vdevx
     2                      + (fy(i) + gravy*rmass(i))*vy(i)
     3                      + (fz(i) + gravz*rmass(i))*vz(i))
      zetad = zetad + rmass(i)*rmass(i)*(vdevx**2+vy(i)**2+vz(i)**2)
 20   continue
      zeta = zetan/(zetad + 1.e-20)
      endif
c
      vdrag = 0.
c
      do 30 i=ind1,ind2
c-----quaternions at start of time step
      q1old(i) = q1new(i)
      q2old(i) = q2new(i)
      q3old(i) = q3new(i)
      q4old(i) = q4new(i)
c
      vdevx = vx(i) - edot*y(i)
      rmasi = 1./rmass(i)
c-----set velocity squared dependent drag force
      if(drag.ne.0.)
     +  vdrag=rmasi*drag*sqrt(vdevx*vdevx+vy(i)*vy(i)+vz(i)*vz(i))
c
c-----velocity increments
      dvx(i) = (fx(i)*rmasi + gravx - (zeta + vdrag)*vdevx)*dt
      dvy(i) = (fy(i)*rmasi + gravy - (zeta + vdrag)*vy(i))*dt
      dvz(i) = (fz(i)*rmasi + gravz - (zeta + vdrag)*vz(i))*dt
 30   continue
c
      if(t.le.0.) then
c-----first time step
         do 40 i=ind1,ind2
c-----trans. velocities at half time step before time zero
         vhx(i) = vx(i) - half*dvx(i)
         vhy(i) = vy(i) - half*dvy(i)
         vhz(i) = vz(i) - half*dvz(i)
 40      continue
      else
c-----other than first time step
         do 50 i=ind1,ind2
c-----trans. velocities at start of current time step
         vx(i) = vhx(i) + half*dvx(i)
         vy(i) = vhy(i) + half*dvy(i)
         vz(i) = vhz(i) + half*dvz(i)
 50      continue
      goto 10
      endif
c
c-----maximum allowed number of iterations reached
 60   continue
c
      if(izeta.eq.2) then
c-----rescale trans. dev. velocities to constant t
      fvsqm = 0.
      f = 0.
      do 70 i=ind1,ind2
         vdevx = vx(i) - edot*y(i)
         fvsqm = fvsqm + rmass(i)*(vdevx**2 + vy(i)**2 + vz(i)**2)
 70   continue
      if(fvsqm.gt.0.) f = sqrt(vsqmz/fvsqm)
      do 80 i=ind1,ind2
         vx(i) = f*(vx(i) - edot*y(i))  +  edot*y(i)
         vy(i) = f*vy(i)
         vz(i) = f*vz(i)
 80   continue
      endif
c
c
      return
      end
c
c ***  *     *  *******  *******   ******   ******
c  *   **    *     *     *         *        *    *
c  *   * *   *     *     *         *             *
c  *   *  *  *     *     ****      *   ***      **
c  *   *   * *     *     *         *     *    **
c  *   *    **     *     *         *     *   *
c ***  *     *     *     *******   ******  *******
c
      subroutine integ2
      include 's3dscmn'
c-----finish this integration step to obtain coordinates at end of
c-----current time step and estimation of velocities there
c
      do 30 i=ind1,ind2
c-----velocities at midpoint of time step
      vhx(i) = vx(i) + half*dvx(i)
      vhy(i) = vy(i) + half*dvy(i)
      vhz(i) = vz(i) + half*dvz(i)
      whx(i) = wx(i) + half*dwx(i)
      why(i) = wy(i) + half*dwy(i)
      whz(i) = wz(i) + half*dwz(i)
c
c-----change in coordinates
      dx(i) = vhx(i)*dt
      dy(i) = vhy(i)*dt
      dz(i) = vhz(i)*dt
c
c-----coordinates at end of time step
      x(i) = x(i) + dx(i)
      y(i) = y(i) + dy(i)
      z(i) = z(i) + dz(i)
c
c-----estimated velocities at end of time step
      vx(i) = vhx(i) + half*dvx(i)
      vy(i) = vhy(i) + half*dvy(i)
      vz(i) = vhz(i) + half*dvz(i)
      wx(i) = whx(i) + half*dwx(i)
      wy(i) = why(i) + half*dwy(i)
      wz(i) = whz(i) + half*dwz(i)
c
c-----rotation coefficients
      bx = fourth*dt*whx(i)
      by = fourth*dt*why(i)
      bz = fourth*dt*whz(i)
      bxsq = bx*bx
      bysq = by*by
      bzsq = bz*bz
```

```
c
c-----determinant of coefficients of quaternion equations
      detb = 1. + two*(bxsq + bysq + bzsq
     1              + bxsq*bysq + bysq*bzsq + bzsq*bxsq)
     2          + bxsq*bxsq + bysq*bysq + bzsq*bzsq
c
c-----constant factor for solution of quaternion equations
      con = (1. + bxsq + bysq + bzsq)/detb
c
c-----right-hand side
      bb1 =    q1old(i) + bz*q2old(i) - bx*q3old(i) - by*q4old(i)
      bb2 = -bz*q1old(i) +   q2old(i) - by*q3old(i) + bx*q4old(i)
      bb3 =  bx*q1old(i) + by*q2old(i) +   q3old(i) + bz*q4old(i)
      bb4 =  by*q1old(i) - bx*q2old(i) - bz*q3old(i) +   q4old(i)
c
c-----new quaternions
      q1new(i) = con*( bb1    + bb2*bz - bb3*bx - bb4*by)
      q2new(i) = con*(-bb1*bz + bb2    - bb3*by + bb4*bx)
      q3new(i) = con*( bb1*bx + bb2*by + bb3    + bb4*bz)
      q4new(i) = con*( bb1*by - bb2*bx - bb3*bz + bb4   )
c
c-----scale factor
      factor = sqrt(1./(q1new(i)*q1new(i) + q2new(i)*q2new(i)
     1              + q3new(i)*q3new(i) + q4new(i)*q4new(i)))
c
c-----rescale quaternions
      q1new(i) = factor*q1new(i)
      q2new(i) = factor*q2new(i)
      q3new(i) = factor*q3new(i)
      q4new(i) = factor*q4new(i)
 30   continue
c
c-----calculate new x coordinates of xz-boundary particles at y = zero
      do 40 i=ind2+1,ind2+nby0
         dx(i) = vx(i)*dt
         x(i) = x(i) + dx(i)
 40   continue
c
c-----calculate new x coordinates of xz-boundary particles at y = ycell
      do 50 i=ind2+nby0+1,ind2+nby0+nby1
         dx(i) = vx(i)*dt
         x(i) = x(i) + dx(i)
 50   continue
c
c-----increment time
      nstep = nstep + 1
      t = nstep*dt
c
c-----increment shear shift parameters
      nshift = nshift + 1
      tshift = nshift*dt
      if(nshift.ge.mshift) then
         tshift = 0.
         nshift = 0
      endif
c
c-----test for move of particles back to primary cell
      imove = 0
      do 60 i=1,np
         if(abs(x(i)).gt.xmax*xcell) imove = 1
 60   continue
c
      if(imove.gt.0) then
c-----move particles back to primary cell and adjust x velocity
         if(ireal.eq.0) then
c--------y-coordinate in primary cell
            do 70 i=1,np
               yshift = ycell*int(y(i)*ycelli)
               if(y(i).lt.0.) yshift = yshift - ycell
               y(i) = y(i) - yshift
c--------shear correction to x-corrdinate
               x(i) = x(i) - yshift*(edot*tshift - int(edot*tshift))
c--------shear correction to x-velocity
               vhx(i) = vhx(i) - yshift*edot
               vx(i) = vhx(i) + half*dvx(i)
c--------shear correction to dx(i)
               dx(i) = vhx(i)*dt
 70         continue
         endif
c
c-----shift x-coordinate into primary cell
         do 80 i=1,np
            x(i) = x(i) - xcell*int(x(i)*xcelli)
            if(x(i).lt.0.) x(i) = x(i) + xcell
 80      continue
c
         if(ireal.lt.2) then
c-----shift z-coordinate into primary cell
         do 85 i=1,np
            z(i) = z(i) - zcell*int(z(i)*zcelli)
            if(z(i).lt.0.) z(i) = z(i) + zcell
 85      continue
         endif
      endif
c
c-----find max displacement in this timestep (for update check)
      dmax2 = 0.
      do 90 i=1,np
         dsqrd = dx(i)*dx(i) + dy(i)*dy(i) + dz(i)*dz(i)
         if(dsqrd.gt.dmax2) dmax2 = dsqrd
  90  continue
      drmax = sqrt(dmax2)
c
      return
      end
c
c  *****    *****   ******   ******   *****   *     *  *******
c *     *        *   *    *  *        *     *  *  *  *  *     *
c *     *        *   *    *  *        *     *  *  *  *  *     *
c *     *   ****     *    *  *****    *     *  *  *  *  *     *
c *     *  *         *    *  *        *     *  *  *  *  *     *
c *     *  *         *    *  *        *     *  *  *  *  *     *
c  *****   *******  ******   ******   *****    **  **   *****
c
      subroutine o3dsout(ifirst)
      include 's3dscmn'
      real etime, tarray(2)
      character*8 labw

c-----computer time used
c     call timeused(icpu,io,isys)
c     ttot = (icpu+io+isys)/1000000.
```

```
      ttot = etime(tarray)
      ttot = tarray(1) + tarray(2)
c
      if(ifirst.eq.0) then
c-----write run information to file o3ds
        open(3,file='o3ds',status='unknown')
c
        write(3,351) (title(i),i=1,10)
        write(3,352) version
        if(istart.ne.0) write(3,351) (oldtl(i),i=1,10)
        write(3,353) np,nxby0,nzby0,nxby1,nzby1,nxbz0,nybz0,nxbz1,nybz1
     &     ,nfix,nfxpl,nzcyl,ncut,ncutx,ncuty,nczero,ncmax,ntcol,nvel
        write(3,354) ndump,nyzone,itervm,ireal,imirr,izeta,icoord,iquat
     &     ,itty,ihertz,istart,ialtk,ixyz
        write(3,355) tmax,dt,dtout,dtoutx,dtouty,dtdump,tzero,edot,epsv
     &     ,xcell,ycell,zcell,xyrat,zyrat,pack
     &     ,vavez,vseed
        write(3,370) wxzero,wyzero,wzzero,wzcyl
        write(3,356) vxzero,vyzero,vzzero,sknl,sknlb,elast,elastb
     &     ,slope,slopeb,dashl,dash2,ratk,fmu,fmub,drag
        write(3,357) power,rmassz,restart
     &     ,gravx,gravy,gravz,vxby0,vxby1,vyby0,vyby1,vzby0
     &     ,vzby1
        write(3,358) search,xmax,draddt,ystart,ystop
     &     ,(number(i),i=1,ngroup),(radius(i),i=1,ngroup)
     &     ,(pmass(i),i=1,ngroup)
        write(3,359) finis
        write(3,360)(i,rad(i),i=1,np)
        write(3,361)(i,rmass(i),i=1,np)
        write(3,362)(i,rmnt(i),i=1,np)
c----close file between writes (to empty buffer & facilitate restarts)
        close(3)

        return
      elseif(ifirst.eq.2) then
        write(3,363) ttot/60.
        close(3)
        return
      endif

c
c-----increment time for o3dsout
      tout = tout + dtout
c
c-----check for restart of cumulative averages
      if((tout.gt.tzero).and.(izero.eq.0)) then
c-----reset flag for restart of cumulative average
        izero = 1
c-----estimated collision frequency
        colfrq = three*pack*sqrt((vsqmt+vsqm)/(masst+mass))/rave
        if((ncmax.gt.0).and.(colfrq.gt.0.)) tmax = ncmax/colfrq
c-----initialize cumulative averages
        call initcum2
      endif
c
      savei = 1./save
      massi = 1./mass
c
      riterv = itervs*savei
c
c
c-----long-term cumulative averages
c
c-----for cell:
      savet = savet + save
      masst = masst + mass
      vsqmt = vsqmt + vsqm
      xmcmt = xmcmt + xmcm
      spnxt = spnxt + spnx
      spnyt = spnyt + spny
      spnzt = spnzt + spnz
      ekint = ekint + ekin
      epott = epott + epot
      erott = erott + erot
      compt = compt + comp
      visct = visct + visc
      do 1 j=1,ngroup
        gekint(j) = gekint(j) + gekin(j)
        gerott(j) = gerott(j) + gerot(j)
        gfloxt(j) = gfloxt(j) + gflox(j)
        gfloyt(j) = gfloyt(j) + gfloy(j)
        gflozt(j) = gflozt(j) + gfloz(j)
 1    continue
      do 5 j=1,9
        pnnkt(j) = pnnkt(j) + pnnk(j)
        pnnpt(j) = pnnpt(j) + pnnp(j)
 5    continue
c
      savei = 1./savet
      massi = 1./masst
      binfi  = 1./(savet*(np-nby0-nby1))
      axzi   = 1./(xcell*ycell)
      labw   = "y=0 wall"
c
      do 10 i=1,nvel
        binvxt(i) = binvxt(i) + binvx(i)
        binvyt(i) = binvyt(i) + binvy(i)
        binvzt(i) = binvzt(i) + binvz(i)
        binwxt(i) = binwxt(i) + binwx(i)
        binwyt(i) = binwyt(i) + binwy(i)
        binwzt(i) = binwzt(i) + binwz(i)
      do 10 j=1,ngroup
        gbinvxt(i,j) = gbinvxt(i,j) + gbinvx(i,j)
        gbinvyt(i,j) = gbinvyt(i,j) + gbinvy(i,j)
        gbinvzt(i,j) = gbinvzt(i,j) + gbinvz(i,j)
 10   continue
c
c-----for lower y-wall particles
      pxwalt = pxwalt + pxwall
      pywalt = pywalt + pywall
      pzwalt = pzwalt + pzwall
c
c-----for y zones:
      do 20 i=1,nyzone
        ymasst(i) = ymasst(i) + ymass(i)
        yvsqmt(i) = yvsqmt(i) + yvsqm(i)
        yxmcmt(i) = yxmcmt(i) + yxmcm(i)
        yspnxt(i) = yspnxt(i) + yspnx(i)
        yspnyt(i) = yspnyt(i) + yspny(i)
        yspnzt(i) = yspnzt(i) + yspnz(i)
        yekint(i) = yekint(i) + yekin(i)
        yepott(i) = yepott(i) + yepot(i)
```

```
          yerott(i) = yerott(i) + yerot(i)
          ypackt(i) = ypackt(i) + ypack(i)
          ycompt(i) = ycompt(i) + ycomp(i)
          yvisct(i) = yvisct(i) + yvisc(i)
          yedott(i) = yedott(i) + yedot(i)
 20     continue
        do 25 j=1,9
        do 25 i=1,nyzone
          ypnnkt(i,j) = ypnnkt(i,j) + ypnnk(i,j)
          ypnnpt(i,j) = ypnnpt(i,j) + ypnnp(i,j)
 25     continue
c
c-----move x,y,z to primary cell if specified by icoord
        call coord
c
c-----rms deviatoric velocity for cell
c-----short-term average
        vave  = sqrt(vsqr*massi)
        vavei = 0.
        if(vave.gt.0.) vavei = 1./vave
c-----long-term average
        vavet = sqrt(vsqrt*masst)
        vaveti = 0.
        if(vavet.gt.0.) vaveti = 1./vavet
c
c-----rms deviatoric velocity for y zones
        do 70 i=1,nyzone
          if(ymass(i).gt.0.) then
            ymassi(i) = 1./ymass(i)
            yvave(i)  = sqrt(yvsqr(i)*ymassi(i))
            yvavei(i) = 1./yvave(i)
          else
            ymassi(i) = 0.
            yvave(i)  = 0.
            yvavei(i) = 0.
          endif
c
          if(ymasst(i).gt.0.) then
            ymassti(i) = 1./ymasst(i)
            yvavet(i) = sqrt(yvsqrt(i)*ymassti(i))
            yvaveti(i) = 1./yvavet(i)
          else
            ymassti(i) = 0.
            yvavet(i)  = 0.
            yvaveti(i) = 0.
          endif
 70     continue
c
c-----print diagnostics
        open(3,file='o3ds', status='old', access='append')
        write(3,304) t,ttot

        write(3,313) riterv
c
        write(3,328) search,tupdtl
c
        write(3,318) (labz(i),yvave(i),yvavet(i)
     1                    ,i=1,nyzone)
        write(3,318) labcell,vave,vavet
c
        write(3,334) (labz(i),yxmom(i)*ymassi(i),yxmomt(i)*ymassti(i)
     1                    ,i=1,nyzone)
        write(3,334) labcell,xmom*massi,xmomt*massti
c
        write(3,317) (labz(i),yspnx(i)*ymassi(i),yspnxt(i)*ymassti(i)
     1                    ,i=1,nyzone)
        write(3,317) labcell,spnx*massi,spnxt*massti
c
        write(3,319) (labz(i),yspny(i)*ymassi(i),yspnyt(i)*ymassti(i)
     1                    ,i=1,nyzone)
        write(3,319) labcell,spny*massi,spnyt*massti
c
        write(3,321) (labz(i),yspnz(i)*ymassi(i),yspnzt(i)*ymassti(i)
     1                    ,i=1,nyzone)
        write(3,321) labcell,spnz*massi,spnzt*massti
c
        write(3,333) (labz(i),yedot(i)*savei,yedott(i)*saveti
     1                    ,i=1,nyzone)
c
        write(3,332) (labz(i),ypack(i)*savei,ypackt(i)*saveti
     1                    ,i=1,nyzone)
c
        write(3,305) (labz(i),yekin(i)*savei,yekint(i)*saveti
     1                    ,i=1,nyzone)
        write(3,305) labcell,ekin*savei,ekint*saveti
        if(ngroup.gt.1) write(3,305) (labg(i),gekin(i)*savei
     1                    ,gekint(i)*saveti,i=1,ngroup)
c
        write(3,306) (labz(i),yepot(i)*savei,yepott(i)*saveti
     1                    ,i=1,nyzone)
        write(3,306) labcell,epot*savei,epott*saveti
c
        write(3,307) (labz(i),yerot(i)*savei,yerott(i)*saveti
     1                    ,i=1,nyzone)
        write(3,307) labcell,erot*savei,erott*saveti
        if(ngroup.gt.1) write(3,307) (labg(i),gerot(i)*savei
     1                    ,gerott(i)*saveti,i=1,ngroup)
c
        write(3,308) (labz(i),(yekin(i)+yepot(i)+yerot(i))*savei
     1                    ,(yekint(i)+ypott(i)+yerott(i))*saveti
     2                    ,i=1,nyzone)
        write(3,308) labcell,(ekin+epot+erot)*savei
     1                  ,(ekint+epott+erott)*saveti
c
        write(3,314) (labz(i),ycomp(i)*savei,ycompt(i)*saveti
     1                    ,i=1,nyzone)
        write(3,314) labcell,comp*savei,compt*saveti
c
        write(3,315) (labz(i),yvisc(i)*savei,yvisct(i)*saveti
     1                    ,i=1,nyzone)
        write(3,315) labcell,visc*savei,visct*saveti
c
        write(3,316) (labz(i),sigma*yedot(i)*savei*yvavei(i)
     1                    ,sigma*yedott(i)*saveti*yvaveti(i)
     2                    ,i=1,nyzone)
        write(3,316) labcell,sigma*edot*vavei
     1                  ,sigma*edot*vaveti
c
c
        do 80 j=1,9
        write(3,309) (j,labz(i),ypnnkt(i,j)*savei,ypnnkt(i,j)*saveti
     1                    ,i=1,nyzone)
```

```
      write(3,309) j,labcell,pnnk(j)*savei,pnnkt(j)*saveti
 80   continue
c
c
      do 85 j=1,9
      write(3,320) (j,labz(i),ypnnp(i,j)*savei,ypnnpt(i,j)*saveti
     1                        ,i=1,nyzone)
      write(3,320) j,labcell,pnnp(j)*savei,pnnpt(j)*saveti
 85   continue
c
c
      do 90 j=1,9
      write(3,324) (j,labz(i),(ypnnk(i,j)+ypnnp(i,j))*savei
     1                       ,(ypnnkt(i,j)+ypnnpt(i,j))*saveti
     1                       ,i=1,nyzone)
      write(3,324) j,labcell,(pnnk(j)+pnnp(j))*savei
     1                      ,(pnnkt(j)+pnnpt(j))*saveti
 90   continue
c
c
c----- print stresses on lower y-wall particles
      if(nby0.ne.0)
     1   write(3,341) labw,pxwall*axzi*savei,pxwalt*axzi*saveti,
     1                labw,pywall*axzi*savei,pywalt*axzi*saveti,
     2                labw,pzwall*axzi*savei,pzwalt*axzi*saveti
c
c
c-----print mass flow rates in x, y, and z directions
      write(3,325) (labg(i),gflox(i)*savei
     1                      ,gfloxt(i)*saveti,i=1,ngroup)
      write(3,326) (labg(i),gfloy(i)*savei
     1                      ,gfloyt(i)*saveti,i=1,ngroup)
      write(3,327) (labg(i),gfloz(i)*savei
     1                      ,gflozt(i)*saveti,i=1,ngroup)
c
c
c-----print coordinates
      write(3,312) (i,xp(i),yp(i),zp(i),
     1   q1old(i),q2old(i),q3old(i),q4old(i),i=1,np)
c
c-----print deviatoric translational and rotational velocities
      write(3,322) (i,vx(i)-edot*y(i)-vxave,vy(i)-vyave,vz(i)-vzave
     1   ,wx(i),wy(i),wz(i)-wzave,i=1,np)
c
c-----print translational velocity distributions
      write(3,329) -nvelhalf*delvx,nvelhalf*delvx
     1   ,nvel,delvx,(binvxt(i)*binfi,i=1,nvel)
      write(3,330) -nvelhalf*delvy,nvelhalf*delvy
     1   ,nvel,delvy,(binvyt(i)*binfi,i=1,nvel)
      write(3,331) -nvelhalf*delvz,nvelhalf*delvz
     1   ,nvel,delvz,(binvzt(i)*binfi,i=1,nvel)
c
c-----print rotational velocity distributions
      write(3,335) -nvelhalf*delwx,nvelhalf*delwx
     1   ,nvel,delwx,(binwxt(i)*binfi,i=1,nvel)
      write(3,336) -nvelhalf*delwy,nvelhalf*delwy
     1   ,nvel,delwy,(binwyt(i)*binfi,i=1,nvel)
      write(3,337) -nvelhalf*delwz,nvelhalf*delwz
     1   ,nvel,delwz,(binwzt(i)*binfi,i=1,nvel)
c
      do 100 j=1,ngroup

c-----print translational velocity distributions for size groups
      write(3,338) labg(j),-nvelhalf*delvx,nvelhalf*delvx
     1   ,nvel,delvx,(gbinvxt(i,j)*binfi,i=1,nvel)
      write(3,339) labg(j),-nvelhalf*delvy,nvelhalf*delvy
     1   ,nvel,delvy,(gbinvyt(i,j)*binfi,i=1,nvel)
      write(3,340) labg(j),-nvelhalf*delvz,nvelhalf*delvz
     1   ,nvel,delvz,(gbinvzt(i,j)*binfi,i=1,nvel)
 100  continue
c
      if(itty.eq.1) then
        write(59,501) t,ekin*savei,ekint*saveti
     1    ,epot*savei,epott*saveti,erot*savei,erott*saveti
     2    ,(ekin+epot+erot)*savei,(ekint+epott+erott)*saveti
     3    ,comp*savei,compt*saveti,visc*savei,visct*saveti
          read(59,502) istop
      endif
c
      close(3)
c-----initialize short-term averages
      call initcum1
c
      return
c

 351  format(1x,10a8)
 352  format(1x,'s3dshear version ',a8)
 353  format(
     & " np =      ",i5,5x," $total particles",/,
     & " nxby0 =   ",i5,5x," $bnd. particles in x-dir. at y = zero",/,
     & " nzby0 =   ",i5,5x," $bnd. particles in z-dir. at y = zero",/,
     & " nxby1 =   ",i5,5x," $bnd. particles in x-dir. at y = ycell",/,
     & " nzby1 =   ",i5,5x," $bnd. particles in z-dir. at y = ycell",/,
     & " nxbz0 =   ",i5,5x," $bnd. particles in x-dir. at z = zero",/,
     & " nybz0 =   ",i5,5x," $bnd. particles in y-dir. at z = zero",/,
     & " nxbz1 =   ",i5,5x," $bnd. particles in x-dir. at z = zcell",/,
     & " nybz1 =   ",i5,5x," $bnd. particles in y-dir. at z = zcell",/,
     & " nfix =    ",i5,5x," $other fixed particles              ",/,
     & " nfxpl =   ",i5,5x," $number of fixed planes             ",/,
     & " nzcyl =   ",i5,5x," $number of cylinders parallel to z axis",/,
     & " nout =    ",i5,5x," $number of times to write file o3ds",/,
     & " noutx =   ",i5,5x," $number of times to write file o3dxq",/,
     & " noutv =   ",i5,5x," $number of times to write file o3dv",/,
     & " nczero =  ",i5,5x," $est. collisions to restart of cum ave",/,
     & " ncmax =   ",i5,5x," $est. collisions for entire run",/,
     & " ntcol =   ",i5,5x," $time steps during collision",/,
     & " nvel =    ",i5,5x," $number of intervals for vel. distrib.")
 354  format(
     & " ndump =   ",i5,5x," $number of times to write dump file",/,
     & " nyzone =  ",i5,5x," $number of y zones for diagnostics",/,
     & " itervm =  ",i5,5x," $max iterations per time step",/,
     & " ireal =   ",i5,5x," $flag for type of boundaries",/,
     & " ixirr =   ",i5,5x," $flag for mirror boundaries",/,
     & " izeta =   ",i5,5x," $flag for holding kin energy constant",/,
     & " icoord =  ",i5,5x," $flag for coordinates print out",/,
     & " iquat =   ",i5,5x," $flag for rotations print out",/,
     & " itty =    ",i5,5x," $flag for tty interaction",/,
     & " ihertz =  ",i5,5x," $flag for hertz power law",/,
     & " istart =  ",i5,5x," $flag for restart dump",/,
     & " ialtk =   ",i5,5x," $flag for alt. usage of sknlb et al.",/,
     & " ixyz =    ",i5,5x," $flag to read coords, 0=fixed, 1=fixsbound,
     & 2=all")
```

```
355  format(1p,
     & " tmax =   ",e11.4," $max time for run",/,
     & " dt =     ",e11.4," $time step",/,
     & " dtout =  ",e11.4," $time interval for writing file o3ds",/,
     & " dtoutx = ",e11.4," $time interval for writing file o3dxq",/,
     & " dtoutv = ",e11.4," $time interval for writing file o3dv",/,
     & " dtdump = ",e11.4," $time interval for dumping",/,
     & " tzero =  ",e11.4," $start cumulative averaging",/,
     & " edot =   ",e11.4," $strain rate",/,
     & " epsv =   ",e11.4," $rel. error tol. for packing fraction",/,
     & " xcell=   ",e11.4," $x cell length",/,
     & " ycell=   ",e11.4," $y cell length",/,
     & " zcell=   ",e11.4," $z cell length",/,
     & " xyrat=   ",e11.4," $x/y cell length ratio",/,
     & " zyrat=   ",e11.4," $z/y cell length ratio",/,
     & " pack =   ",e11.4," $solid packing fraction",/,
     & " vave =   ",e11.4," $average deviatoric transl. velocity",/,
     & " vseed =  ",e11.4," $seed for random velocities")
370  format(1p,
     & " wxzero = ",e11.4," $initial x- rot. vel. active particles",/,
     & " wyzero = ",e11.4," $initial y- rot. vel. active particles",/,
     & " wzzero = ",e11.4," $initial z- rot. vel. active particles",/,
     & " wzcyl =  ",e11.4," $rotational vel. 1st z-cylinder (bndry)")
356  format(1p,
     & " vxzero = ",e11.4," $initial non-dev. transl. vel. in x-dir.",/,
     & " vyzero = ",e11.4," $initial non-dev. transl. vel. in y-dir.",/,
     & " vzzero = ",e11.4," $initial non-dev. transl. vel. in z-dir.",/,
     & " sknl =   ",e11.4," $normal force coefficient",/,
     & " sknlb =  ",e11.4," $ditto for boundary/fixed particles",/,
     & " elast =  ",e11.4," $coefficient of restitution",/,
     & " elastb = ",e11.4," $ditto for boundary/fixed particles",/,
     & " slope =  ",e11.4," $alternative parameter for unloading",/,
     & " slopeb = ",e11.4," $ditto for boundary/fixed particles",/,
     & " dashl =  ",e11.4," $linear viscous damping coefficient",/,
     & " dash2 =  ",e11.4," $overlap weighted damping coefficient",/,
     & " ratk =   ",e11.4," $ratio of tangential/normal stiffness",/,
     & " fmu =    ",e11.4," $coefficient of friction",/,
     & " fmub =   ",e11.4," $friction for bndary or fixed particles",/,
     & " drag =   ",e11.4," $velocity squared drag force coefficient")
357  format(1p,
     & " power    ",e11.4," $tangential force exponent",/,
     & " rmassz = ",e11.4," $mass of particle having unit radius",/,
     & " tstart = ",e11.4," $time to restart run",/,
     & " gravx =  ",e11.4," $acceleration of gravity in x direction",/,
     & " gravy =  ",e11.4," $acceleration of gravity in y direction",/,
     & " gravz =  ",e11.4," $acceleration of gravity in z direction",/,
     & " vzby0 =  ",e11.4," $x-vel of xz-boundary particle (y=zero)",/,
     & " vxby1 =  ",e11.4," $x-vel of xz-boundary particle (y=ycell)",/,
     & " vyby0 =  ",e11.4," $y-vel of xz-boundary particle (y=zero)",/,
     & " vyby1 =  ",e11.4," $y-vel of xz-boundary particle (y=ycell)",/,
     & " vzby0 =  ",e11.4," $z-vel of xz-boundary particle (y=zero)",/,
     & " vzby1 =  ",e11.4," $z-vel of xz-boundary particle (y=ycell)")
358  format(1p,
     & " search = ",e11.4," $distance for near-neighbor search",/,
     & " xmax =   ",e11.4," $maximum allowable abs(x-coordinate)",/,
     & " draddt = ",e11.4," $rate of increase of particle radii",/,
     & " ystart = ",e11.4," $y-coordinate of bottom of first yzone",/,
     & " ystop  = ",e11.4," $y-coordinate of top of last yzone",/,
     & " number = ",10(i5,5x),/,21x," $particles in each size group",/,
     & " radius = ",10(e10.3),/,21x," $particle radius for group",/,
     & " pmass  = ",10(e10.3),/,21x," $particle mass for group")
c
 359  format(1p,
     & " finis =  ",e11.4," $end")
c
 360  format(1p,/," particle radii",/,
     &      (1x,5(i5,e11.4),/))
c
 361  format(1p,/," particle masses",/,
     &      (1x,5(i5,e11.4),/))
c
 362  format(1p,/," moments of inertia",/,
     &      (1x,5(i5,e11.4),/))
d
 363  format(//,' Computer time used (min) = ',1pe13.7)
c
 304  format(/,1x,"***time = ",1p,e12.4,/,
     1       1x,"cumulative computer time (s) = ",e12.4)
 313  format(1x,"average number of iterations = ",1p,e12.4)
 328  format(1x,"search radius-near neighbors= ",e12.4,/,
     1       1x,"last update time             = ",1p,e12.4)
 318  format(1x,"rms deviatoric vel ",a8," = ",1p,2e12.4)
 334  format(1x,"mean total x vel   ",a8," = ",1p,2e12.4)
 333  format(1x,"strain rate        ",a8," = ",1p,2e12.4)
 332  format(1x,"packing fraction   ",a8," = ",1p,2e12.4)
 305  format(1x,"trans. kin. energy ",a8," = ",1p,2e12.4)
 306  format(1x,"trans. pot. energy ",a8," = ",1p,2e12.4)
 307  format(1x,"rot. kin. energy   ",a8," = ",1p,2e12.4)
 308  format(1x,"total energy       ",a8," = ",1p,2e12.4)
 317  format(1x,"spin ang. mom. (x) ",a8," = ",1p,2e12.4)
 319  format(1x,"spin ang. mom. (y) ",a8," = ",1p,2e12.4)
 321  format(1x,"spin ang. mom. (z) ",a8," = ",1p,2e12.4)
 314  format(1x,"compressibility    ",a8," = ",1p,2e12.4)
 315  format(1x,"viscosity          ",a8," = ",1p,2e12.4)
 316  format(1x,"savage vel. ratio  ",a8," = ",1p,2e12.4)
 309  format(1x,"stress : kin. (",i1,")  ",a8," = ",1p,2e12.4)
 320  format(1x,"stress : pot. (",i1,")  ",a8," = ",1p,2e12.4)
 324  format(1x,"stress : tot. (",i1,")  ",a8," = ",1p,2e12.4)
 325  format(1x,"mass flux (x)      ",a8," = ",1p,2e12.4)
 326  format(1x,"mass flux (y)      ",a8," = ",1p,2e12.4)
 327  format(1x,"mass flux (z)      ",a8," = ",1p,2e12.4)
 312  format(/,3x,"i",2x,"x(i)",6x,"y(i)",6x,"z(i)",6x,"xi(i)"
     1   ,7x,"eta(i)",6x,"zeta(i)",5x,"chi(i)"
     2   ,1p,/,(1x,i4,7e12.4))
 322  format(/,3x,"i",2x,"vx(i)",7x,"vy(i)",7x,"vz(i)",7x
     1   ,"wx(i)",7x,"wy(i)",7x,"wz(i)"
     2   ,1p,/,(1x,i4,6e12.4))
 329  format(1x,"distribution of vx  (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
 330  format(1x,"distribution of vy  (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
 331  format(1x,"distribution of vz  (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
 335  format(1x,"distribution of wx  (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
 336  format(1x,"distribution of wy  (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
 337  format(1x,"distribution of wz  (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
c
 338  format(1x,"distrib ",a8," vx (",1p,e11.4," to ",e11.4,
     1       " in ",i3," steps of ",e11.4,")",/,(1x,10e11.4))
```

```
 339  format(1x,"distrib ",a8," vy (",1p,el1.4," to ",el1.4,
     1       " in ",i3," steps of ",el1.4,")",/,(1x,10el1.4))
 340  format(1x,"distrib ",a8," vz (",1p,el1.4," to ",el1.4,
     1       " in ",i3," steps of ",el1.4,")",/,(1x,10el1.4))
 341  format(1x,"boundary x-stress  ", a8, " = ",1p,2el2.4,/,
     1       1x,"boundary y-stress  ", a8, " = ",2el2.4,/,
     2       1x,"boundary z-stress  ", a8, " = ",2el2.4)
c
 501  format(" time = ",1p,el2.4," ekin  = ",2el2.4,/,
     1                  20x," epot  = ",2el2.4,/,
     2                  20x," erot  = ",2el2.4,/,
     3                  20x," etot  = ",2el2.4,/,
     5                  20x," corp  = ",2el2.4,/,
     6                  20x," visc  = ",2el2.4,/,
     8       " to stop calculation, type -1, otherwise return")
 502  format(i2)
      end


      subroutine coord
      include 's3dscmm'
      if(icoord.eq.0) then
c-----move coordinates back to primary cell for printout

        if(ireal.eq.0) then
c-----y-coordinate in primary cell
          do 20 i=ind1,ind2
            yshift = ycell*int(y(i)*ycell1)
            if(y(i).lt.0.) yshift = yshift - ycell
            yp(i) = y(i) - yshift
c-----shear correction to x-corrdinate
            xp(i) = x(i) - yshift*(edot*tshift - int(edot*tshift))
c-----no shear correction to z-coordinate
            zp(i) = z(i)
 30       continue
c-----no shear corrections for fixed particles
          if(nfix.gt.0).or.(nfxpl.gt.0)) then
            do 35 i=ind1fx,ind2fxp
              xp(i) = x(i)
              yp(i) = y(i)
              zp(i) = z(i)
 35         continue
          endif

        else
c-----no y shift or shear correction for any particles when ireal.ne.0
          do 40 i=1,np
            xp(i) = x(i)
            yp(i) = y(i)
            zp(i) = z(i)
 40       continue
        endif

c-----shift x-coordinate into primary cell
        do 50 i=1,np
          xp(i) = xp(i) - xcell*int(xp(i)*xcell1)
          if(xp(i).lt.0.) xp(i) = xp(i) + xcell
 50     continue

        if(ireal.lt.2) then
c-----shift z-coordinate into primary cell
          do 55 i=1,np
            zp(i) = zp(i) - zcell*int(zp(i)*zcell1)
            if(zp(i).lt.0.) zp(i) = zp(i) + zcell
 55       continue
          endif
        else

c-----do not move coordinates back to primary cell for printout
          do 60 i=1,np
            xp(i) = x(i)
            yp(i) = y(i)
            zp(i) = z(i)
 60       continue
        endif

        return
        end
c
c  *****   *****  ******  *      *  *****  *    *  *******
c *     *       *  *    *  *    *   *      *  *      *
c *     *       *  *    *  *    *   *      * *       *
c *     *  ****   *    *  *    *   *****  **        *
c *     *       *  *    *  *    *   *      * *       *
c *     *       *  *    *   *  *    *      *  *      *
c  *****  *****   ******    **     *****  *    *     *
c
      subroutine o3dvout(ifirst)
      include 's3dscmm'
c

      if(ifirst.eq.0) then
        open(8,file='o3dv',status='unknown')
        write(8,301) (title(i),i=1,10)
        write(8,300) version
        write(8,302) (number(i),i=1,ngroup)
        write(8,303) (radius(i),i=1,ngroup)
      else
         open(8,file='o3dv',status='old', access='append')
        write(8,304) t
c-----write linear velocities and angular velocities of all particles
        write(8,305) (vx(i),vy(i),vz(i),wx(i),wy(i),wz(i),i=1,np)
c-----increment time for writing file o3dv
        toutv = toutv + dtoutv
      endif

      close(8)

      return

 300  format(1x,'s2dshear version ',a8)
 301  format(1x,10a8)
 302  format(1x,'number ',10(1x,i5,5x))
 303  format(1x,'radius ',10(1pe11.3))
 304  format(1x,'  time ',1pe12.4)
 305  format(6(1pe13.5))
      end

c
c  *****   *****  ******  *      *  *****  *    *  *******
c *     *       *  *    *  *    *   *      *  *      *
c *     *       *  *    *  *    *   *      * *       *
c *     *  ****   *    *  *    *   *****  **        *
```

```
c *    *   ****   *   *    *    *  * *    *   *
c *    *       * *  *   *  * *   *  * *    *   *
c *    *       * *  *   *  *  *  *  *  *   *   *
c *****    *****  ******  *    *  *****  *****   *
c
      subroutine o3dxout(ifirst)
      include 's3dscmn'
c-----variables for successive rotation angles about +x, +y, and +z
      real*8 rot(3,3),angx,angy,angz
      if(ifirst.eq.0) then
         open(7,file='o3dx',status='unknown')
        write(7,301) (title(i),i=1,10)
        write(7,300) version
        write(7,302) (number(i),i=1,ngroup)
        write(7,303) (radius(i),i=1,ngroup)
         write(7,306) xcell, ycell, zcell
      else
c--------write out rotations to file o3dx
         open(7,file='o3dx',status='old', access='append')
        write(7,304) t
        fact = 180./pi
        halfpi = 0.5*pi
        twopi = 2.*pi
        do 10 i=1,np
c-----------calculate rotation matrix from quaternions
            call matrot(q1old(i),q2old(i),q3old(i),q4old(i),rot)
c-----------calculate successive rotation angles about +x, +y, and +z
            call angles(rot,angx,angy,angz,halfpi,twopi)
c-----------move x,y,z to primary cell, if specified by icoord
            call coord
c-----------write position and rotation angles
            write(7,305) xp(i),yp(i),zp(i),angx,angy,angz
 10      continue
c--------increment time for writing file o3dx
        toutx = toutx + dtoutx
      endif
      close(7)
      return
 300  format(1x,'s3dshear version ',a8)
 301  format(1x,10a8)
 302  format(1x,'number ',10(1x,i5,5x))
 303  format(1x,'radius ',10(1pe11.3))
 304  format(1x,'  time ',1pe12.4)
 305  format(6(1pe13.5))
 306  format(1x,'xcell ycell zcell ',1p3e13.5)
      end

      subroutine o3dxqout(ifirst)
      include 's3dscmn'
      if(ifirst.eq.0) then
         open(9,file='o3dxq',status='unknown')
         write(9,301) (title(i),i=1,10)
         write(9,300) version
         write(9,302) (number(i),i=1,ngroup)
         write(9,303) (radius(i),i=1,ngroup)
         write(9,306) xcell, ycell, zcell
      else
c--------write out quaternions to file o3dxq
         open(9,file='o3dxq',status='old', access='append')
         write(9,304) t
         do 10 i=1,np
c-----------move x,y,z to primary cell, if specified by icoord
            call coord
c-----------write position and quaternions
            write(9,305) xp(i),yp(i),zp(i),q1old(i),q2old(i),q3old(i),
     >                   q4old(i)
 10      continue
c--------increment time for writing files o3dxq
         toutx = toutx + dtoutx
      endif
      close(9)
      return
 300  format(1x,'s3dshear version ',a8)
 301  format(1x,10a8)
 302  format(1x,'number ',10(1x,i5,5x))
 303  format(1x,'radius ',10(1pe11.3))
 304  format(1x,'  time ',1pe12.4)
 305  format(7(1pe13.5))
 306  format(1x,'xcell ycell zcell ',1p3e13.5)
      end

      subroutine matrot(q1,q2,q3,q4,r)
c-----rotation matrix (r) from quaternions (qi)
      real*8 q1,q2,q3,q4,r(3,3)

      r(1,1) = -q1*q1 + q2*q2 - q3*q3 + q4*q4
      r(2,2) =  q1*q1 - q2*q2 - q3*q3 + q4*q4
      r(3,3) = -q1*q1 - q2*q2 + q3*q3 + q4*q4

      r(2,1) = -2*(q3*q4 + q1*q2)
      r(1,2) =  2*(q3*q4 - q1*q2)

      r(3,1) =  2*(q2*q3 - q1*q4)
      r(1,3) =  2*(q2*q3 + q1*q4)

      r(3,2) = -2*(q2*q4 + q1*q3)
      r(2,3) =  2*(q2*q4 - q1*q3)

      return
      end


      subroutine angles(r,x,y,z,halfpi,twopi)
c-----calculate successive rotation about +x, +y, +z from rotation matrix
      real*8 r(3,3),x,y,z,cx,sx,fx,halfpi,twopi

      if(1.-abs(r(3,1)).lt.1.e-12) then
         x = atan2(r(1,2),r(2,2))
         y = sign(halfpi,r(3,1))
         z = 0.
      else
         x = atan2(-r(3,2),r(3,3))
         z = atan2(-r(2,1),r(1,1))
         cx = cos(x)
         sx = sin(x)
         if(abs(sx).gt.abs(cx)) then
            fx = abs(r(3,2)/sx)
            y = atan2(r(3,1),fx)
         else
            fx = abs(r(3,3)/cx)
            y = atan2(r(3,1),fx)
         endif
```

```
      endif
      x = mod(x+twopi,twopi)
      y = mod(y+twopi,twopi)
      z = mod(z+twopi,twopi)
      return
      end
c
c ******    ***    *****  *   **  *** *    *   ******
c *    *  *   *  *     * * *  **    *   **   * *
c *    * *     * *     ***       *  * *   * *
c *****  ******* *       * *     *  * * * *  ***
c *      *     * *       *  *    *  *  * *    *
c *      *     * *     * *   *   *  *   **  *     *
c *      *     *  *****  *    *  *** *    *   ******
c
      subroutine packing(index1,index2,pak)
      include 's3dscmm'
      real*4 randf1
      real*8 rot(3,3)
      real pak
c-----this subroutine estimates the packing fraction of
c-----boundary particles and/or other fixed particles in cell
c
c-----pick itot random locations
        ivol = 0
      do 10 i=1,itot
        xi = randf1(0.)*xcell
        yi = randf1(0.)*ycell
        zi = randf1(0.)*zcell
c
c-----is the random location within a particle in the specified range
      do 5 j=index1,index2
        rx = x(j) - xi
        ry = y(j) - yi
        rz = z(j) - zi
c
c-----cylinders
        if(j.ge.indlcz) rz = 0.
c-----boundary planes
        if(((j.eq.indly0).and.(nby0.eq.1)).or.
     1     ((j.eq.indly1).and.(nby1.eq.1))) then
           rx = 0.
           rz = 0.
        elseif(((j.eq.indlz0).and.(nbz0.eq.1)).or.
     1          ((j.eq.indlz1).and.(nbz1.eq.1))) then
           rx = 0.
           ry = 0.
        endif
c
c-----find nearest image delta x
        rx = rx - xcell*int(rx*xcelli)
        if(abs(rx).gt.xcellh) rx = rx - sign(xcell,rx)
c
      if(ireal.lt.1) then
c-----find nearest image delta-y
        ry = ry - ycell*int(ry*ycelli)
        if(abs(ry).gt.ycellh) ry = ry - sign(ycell,ry)
      endif
c
      if(ireal.lt.2) then
c-----find nearest image delta z
        rz = rz - zcell*int(rz*zcelli)
        if(abs(rz).gt.zcellh) rz = rz - sign(zcell,rz)
      endif
c
c-----fixed planes
      if((j.ge.indlfxp).and.(j.le.ind2fxp)) then
           xplanh = xplnh(j)
           zplanh = zplnh(j)
c-----rotate coordinate system to make plane horizontal
         call matrot(q1old(j),q2old(j),q3old(j),-q4old(j),rot)
         rxr = rx*rot(1,1) + ry*rot(1,2) + rz*rot(1,3)
         ryr = rx*rot(2,1) + ry*rot(2,2) + rz*rot(2,3)
         rzr = rx*rot(3,1) + ry*rot(3,2) + rz*rot(3,3)
c-----detect borders of plane
         if(abs(rxr).gt.xplanh) then
           rxr = rxr - sign(xplanh,rxr)
         else
           rxr = 0
         endif
         if(abs(rzr).gt.zplanh) then
           rzr = rzr - sign(zplanh,rzr)
         else
           rzr = 0
         endif
c-----rotate back the coordinate system
         call matrot(q1old(j),q2old(j),q3old(j),q4old(j),rot)
         rx = rxr*rot(1,1) + ryr*rot(1,2) + rzr*rot(1,3)
         ry = rxr*rot(2,1) + ryr*rot(2,2) + rzr*rot(2,3)
         rz = rxr*rot(3,1) + ryr*rot(3,2) + rzr*rot(3,3)
      endif
c
         rijsq = rx*rx + ry*ry + rz*rz
c
c****   if(rijsq.le.radz(j)**2) then
      if(sign(rijsq,radz(j)).le.sign(radz(j)**2,radz(j))) then
        ivol = ivol + 1
        goto 10
      endif
c
 5    continue
c
 10   continue
c
c-----estimated packing fraction of these particles
      pak = float(ivol)/float(itot)
      return
      end
c
c ******    ***    *     *  ******
c *    *  *   *  *     *  *  *    *
c *    * *     * *     *  *  *    *
c *****  ******* *     *  *  *    *
c *  *   *     *  *   *   *  *    *
c *   *  *     *  *   *   **  *    *
c *    * *     *   * *    *  ******
c
      function randf1 (r)
c***begin prologue  randf1
c***revision date  811015  (yymmdd)
c***category no.  g5e
c***keywords random number, uniform, special function
```

```
c***date written  april 1977
c***author  fullerton w. (lasl)
c***purpose
c generates a uniformly distributed random number.
c***description
c april 1977 version.  w. fullerton, c3, los alamos scientific lab.
c
c      this pseudo-random number generator is portable among a wide
c variety of computers.  randf1(r) undoubtedly is not as good as many
c readily available installation dependent versions, and so this
c routine is not recommended for widespread usage.  its redeeming
c feature is that the exact same random numbers (to within final round-
c off error) can be generated from machine to machine.  thus, programs
c that make use of random numbers can be easily transported to and
c checked in a new environment.
c      the random numbers are generated by the linear congruential
c method described, e.g., by knuth in seminumerical methods (p.9),
c addison-wesley, 1969.  given the i-th number of a pseudo-random
c sequence, the i+1 -st number is generated from
c             x(i+1) = (a*x(i) + c) mod m,
c where here m = 2**22 = 4194304, c = 1731 and several suitable values
c of the multiplier a are discussed below.  both the multiplier a and
c random number x are represented in double precision as two 11-bit
c words.  the constants are chosen so that the period is the maximum
c possible, 4194304.
c      in order that the same numbers be generated from machine to
c machine, it is necessary that 23-bit integers be reducible modulo
c 2**11 exactly, that 23-bit integers be added exactly, and that 11-bit
c integers be multiplied exactly.  furthermore, if the restart option
c is used (where r is between 0 and 1), then the product r*2**22 =
c r*4194304 must be correct to the nearest integer.
c      the first four random numbers should be .0004127026,
c .6750836372, .1614754200, and .9086198807.  the tenth random number
c is .5527787209, and the hundredth is .3600893021 .  the thousandth
c number should be .2176990509 .
c      in order to generate several effectively independent sequences
c with the same generator, it is necessary to know the random number
c for several widely spaced calls.  the i-th random number times 2**22,
c where i=k*p/8 and p is the period of the sequence (p = 2**22), is
c still of the form l*p/8.  in particular we find the i-th random
c number multiplied by 2**22 is given by
c l  =  0  1*p/8  2*p/8  3*p/8  4*p/8  5*p/8  6*p/8  7*p/8  8*p/8
c randf1=  0  5*p/8  2*p/8  7*p/8  4*p/8  1*p/8  6*p/8  3*p/8  0
c thus the 4*p/8 = 2097152 random number is 2097152/2**22.
c      several multipliers have been subjected to the spectral test
c (see knuth, p. 82).  four suitable multipliers roughly in order of
c goodness according to the spectral test are
c    3146757 = 1536*2048 + 1029 = 2**21 + 2**20 + 2**10 + 5
c    2098181 = 1024*2048 + 1029 = 2**21 + 2**10 + 5
c    3146245 = 1536*2048 +  517 = 2**21 + 2**20 + 2**9 + 5
c    2776669 = 1355*2048 + 1629 = 5**9 + 7**7 + 1
c
c      in the table below log10(nu(i)) gives roughly the number of
c random decimal digits in the random numbers considered i at a time.
c c is the primary measure of goodness.  in both cases bigger is better.
c
c                     log10 nu(i)              c(i)
c       a       i=2  i=3  i=4  i=5    i=2  i=3  i=4  i=5
c
c    3146757    3.3  2.0  1.6  1.3    3.1  1.3  4.6  2.6
c    2098181    3.3  2.0  1.6  1.2    3.2  1.3  4.6  1.7
c    3146245    3.3  2.2  1.5  1.1    3.2  4.2  1.1  0.4
c    2776669    3.3  2.1  1.6  1.3    2.5  2.0  1.9  2.6
c  best
c    possible   3.3  2.3  1.7  1.4    3.6  5.9  9.7  14.9
c
c            input argument --
c r      if r=0., the next random number of the sequence is generated.
c        if r.lt.0., the last generated number will be returned for
c          possible use in a restart procedure.
c        if r.gt.0., the sequence of random numbers will start with the
c          seed r mod 1.  this seed is also returned as the value of
c          randf1 provided the arithmetic is done exactly.
c
c            output value --
c randf1   a pseudo-random number between 0. and 1.
c
c ial and ia0 are the hi and lo parts of a.  ialma0 = ial - ia0.
c
c***references
c***routines called  (none)
c***end prologue  randf1
      data ial, ia0, ialma0 /1536, 1029, 507/
      data ic /1731/
      data ix1, ix0 /0, 0/
c***first executable statement  randf1
      if (r.lt.0.) goto 10
      if (r.gt.0.) goto 20
c
c          a*x = 2**22*ial*ix1 + 2**11*(ial*ix1 + (ial-ia0)*(ix0-ix1)
c                + ia0*ix0) + ia0*ix0
c
      iy0 = ia0*ix0
      iy1 = ial*ix1 + ialma0*(ix0-ix1) + iy0
      iy0 = iy0 + ic
      ix0 = mod (iy0, 2048)
      iy1 = iy1 + (iy0-ix0)/2048
      ix1 = mod (iy1, 2048)
c
   10 randf1 = ix1*2048 + ix0
      randf1 = randf1 / 4194304.
      return
c
   20 ix1 = amod(r,1.)*4194304. + 0.5
      ix0 = mod (ix1, 2048)
      ix1 = (ix1-ix0)/2048
      goto 10
c
      end
c
c *     *  ******  ******     ***    *******  *******
c *     *  *    *  *    *  *   *  *  *        *
c *     *  *    *  *    *  *     *  *        *
c *     *  *****   *    *  *     *  *******  ****
c *     *  *       *    *  *     *  *        *
c *     *  *       *    *  *     *  *        *
c  *****   *       ******  *     *  *        *******
c
      subroutine update
      include 's3dscmn'
      real*4 randf1
      real*8 rot(3,3)
```

```
c     this subroutine checks all particle pairs and updates the near-
c      neighbor arrays for all active particles
c
 10   do 500 i=imin,imax
      do 500 j=i+1,np
c
         rsum = rad(i) + rad(j)
          rsum2 = rsum*rsum
c
c-----delta x and delta y
         rx = x(j) - x(i)
         ry = y(j) - y(i)
         rz = z(j) - z(i)
c
c-----cylinders
         if(j.ge.indlcz) rz = 0.
c-----boundary planes
         if(((j.eq.indly0).and.(nby0.eq.1)).or.
     1       ((j.eq.indly1).and.(nby1.eq.1))) then
            rx = 0.
            rz = 0.
            if((j.eq.indly1).and.(imirr.eq.1)) then
c-----mirror on y=ycell
               rsum = rad(i) + rad(i)
               rsum2 = rsum*rsum
               ry = 2.*(ry - rad(j)*brown(i))
            endif
         elseif(((j.eq.indlz0).and.(nbz0.eq.1)).or.
     1          ((j.eq.indlz1).and.(nbz1.eq.1))) then
            rx = 0.
            ry = 0.
         endif
c
       if(ireal.eq.0) then
c-----find nearest image delta-y
         inty = (ry + ycellh)*ycelli + 10.
          yshift = inty - 10
          ry = ry - yshift*ycell
c-----shear correction to delta x
         rx = rx - yshift*(edot*tshift - int(edot*tshift))
       endif
c
c-----find nearest image delta x
          intx = (rx + xcellh)*xcelli + 10.
          intx = intx - 10
          rx = rx -intx*xcell
c
       if(ireal.lt.2) then
c-----find nearest image delta z
          intz = (rz + zcellh)*zcelli + 10.
          intz = intz - 10
          rz = rz - intz*zcell
       endif
c
c-----fixed planes
       if((j.ge.indlfxp).and.(j.le.ind2fxp)) then
            xplanh = xplnh(j)
            zplanh = zplnh(j)
c-----rotate coordinate system to make plane horizontal
          call matrot(q1old(j),q2old(j),q3old(j),-q4old(j),rot)
          rxr = rx*rot(1,1) + ry*rot(1,2) + rz*rot(1,3)
          ryr = rx*rot(2,1) + ry*rot(2,2) + rz*rot(2,3)
          rzr = rx*rot(3,1) + ry*rot(3,2) + rz*rot(3,3)
c-----detect borders of plane
          if(abs(rxr).gt.xplanh) then
             rxr = rxr - sign(xplanh,rxr)
          else
             rxr = 0
          endif
          if(abs(rzr).gt.zplanh) then
            rzr = rzr - sign(zplanh,rzr)
          else
             rzr = 0
          endif
c-----rotate back the coordinate system
          call matrot(q1old(j),q2old(j),q3old(j),q4old(j),rot)
          rx = rxr*rot(1,1) + ryr*rot(1,2) + rzr*rot(1,3)
          ry = rxr*rot(2,1) + ryr*rot(2,2) + rzr*rot(2,3)
          rz = rxr*rot(3,1) + ryr*rot(3,2) + rzr*rot(3,3)
       endif
c
       rijsq = rx*rx + ry*ry + rz*rz
       rnear2 =(rsum + search)**2
c
c****     if(rijsq.gt.rnear2) goto 500
      if(((rad(j).gt.0.).and.(rijsq.gt.rnear2)).or.
     >   ((rad(j).lt.0.).and.(rijsq.lt.rnear2))) go to 500
c
c-----check to see if j is already in i's list
       jdx = nebor(i)
       if(jdx.eq.0) goto 200
       idx = i2or1*jdx - i1or0
  100 jj=ndx(idx)
       if(j.eq.jj) goto 500
       if(next(idx).eq.0) goto 220
       jdx = next(idx)
       idx = i2or1*jdx - i1or0
       goto 100
c
c---- this will be first entry in i's list
c----     if((rijsq.le.rsum2).and.(t.gt.0.).and.(iextra.ge.100))goto 700
  200 if(((rad(j).gt.0.).and.(rijsq.lt.rsum2).and.(t.gt.0.).and.
     1    (iextra.ge.100)).or.
     2   ((rad(j).lt.0.).and.(rijsq.gt.rsum2).and.(t.gt.0.).and.
     3    (iextra.ge.100)))  goto 700
c
       jdxnew = mt1
       idxnew = i2or1*jdxnew - i1or0
       nebor(i) = jdxnew
       goto 300
c
c----- add this entry to end of i's list
c-----     if((rijsq.le.rsum2).and.(t.gt.0.).and.(iextra.ge.100))goto 700
  220 if(((rad(j).gt.0.).and.(rijsq.lt.rsum2).and.(t.gt.0.).and.
     1    (iextra.ge.100)).or.
     2   ((rad(j).lt.0.).and.(rijsq.gt.rsum2).and.(t.gt.0.).and.
     3    (iextra.ge.100)))  goto 700
c
       jdxnew = mt1
       idxnew = i2or1*jdxnew - i1or0
       next(idx) = jdxnew
  300 imt1 = i2or1*mt1 - i1or0
```

```
      if(next(imtl).eq.0) then
c-----  utilize previously unused linked-list storage space
        mtl = mtl + nwd
        if(mtl.gt.maxwd) goto 600
        imtl = i2or1*mtl - i1or0
         next(imtl) = 0
c------ set llused to the largest index ever used in linked list
         llused = mtl + nwd
         illused = i2or1*llused - i1or0
      else
c------ use previously released storage in 'empty' (i.e. 'mt' list)
        mtl = next(imtl)
      endif
c
c-----initialize entries in linked list for this neighbor
      ndx(idxnew) = j
      next(idxnew)  = 0
      a(jdxnew)   = 0.
      a0(jdxnew)  = 0.
      fn(jdxnew)  = 0.
      tfx(jdxnew) = 0.
      tfy(jdxnew) = 0.
      tfz(jdxnew) = 0.
      tm(jdxnew)  = 0.
      if((j.eq.indly1).and.(imirr.eq.1)) brown(i) = 1. + randf1(0.)
  500 continue
c
c-----reset parameters for testing when update is needed
      tupdt1 = t
      delrup = 0.
      return
c
c-----storage is full, reduce search and update time by 10%
  600 search = search * .90
c       retrieve last storage location in available memory
        mtl = mtl - nwd
        if(jdxnew.eq.next(idx)) next(idx) = 0
        if(jdxnew.eq.nebor(i))nebor(i) = 0
      if(drmax.gt.half*search) then
         open (3,file='o3ds',status='unknown', access='append')
        write(3,1000)
        write(*,1000)
        call exit(1)
      endif
      call deletem
      goto 10
c
c-----error near neighbor found after overlap occured - call exit
  700 open (3,file='o3ds',status='unknown', access='append')
      write (3,1700)i,j,i,rad(i),x(i),y(i),z(i),j,rad(j),x(j),y(j),z(j)
     > ,rx,ry,rz,t,search,tupdt1,drmax,delrup,rnear2,rijsq,rsum
      write (*,1700)i,j,i,rad(i),x(i),y(i),z(i),j,rad(j),x(j),y(j),z(j)
     > ,rx,ry,rz,t,search,tupdt1,drmax,delrup,rnear2,rijsq,rsum
      call exit(1)

c-----ran out of particles
  800 open (3,file='o3ds',status='unknown',access='append')
      write (3,fmt="('ran out of particles, time: ',e10.3)")t
      write (*,fmt="('ran out of particles, time: ',e10.3)")t
      call exit(1)
c

 1000 format(" error - attempting to set search less than max",
     >  " displacement in last timestep")
 1700 format(" error exit from subroutine update - ",/,
     1 " near neighbor found after overlap, particles - ",i5," and",i6,
     2 /," no. ",4x,"rad",8x,"x",11x,"y",11x,"z",
     3 /, 1x,i5,1p,4e12.4,/, 1x,i5,4e12.4,/," rx,ry,rz =",7x,3e12.4,
     5 /," time=",e12.4,"  search=",e12.4,"  tupdt1=",e12.4,
     6 /," drmax=",e12.4,"  delrup=",e12.4,"  rnear2=",e12.4,
     7 /," rijsq=",e12.4,"  rsum=",e12.4)
c
      end
```

Archivos de entrada

Corrida 1. Simetría en el eje vertical

```
i3ds001 35deg silo e=0.85, k=2000, fmu=0.5, fmub=0.65
 $var  np =          7004     $end total number of particles in cell
 $var  nfix =          0      $end number of fixed particles (lower boundary on
plane)
 $var  nout =          0      $end number of times to print out results
 $var  noutv =         0      $end number of times to print out velocities
 $var  nczero =        0      $end number of collisions before start cum.  ave.
 $var  ncmax =         0      $end number of collisions during entire run
 $var  ntcol =        40      $end number of timesteps during a collision.
 $var  nyzone =        1      $end number of y zones
 $var  nvel  =         1      $end number of velocity intervals
 $var  itervm =        1      $end max iterations per time step
 $var  ireal =         1      $end flag for periodic boundaries
 $var  imirr =         1      $end flag for mirror boundaries
 $var  nxby0 =         1      $end no. boundary particles in x-dir. at y = zero
 $var  nzby0 =         1      $end no. boundary particles in z-dir. at y = zero
 $var  nxby1 =         1      $end no. boundary particles in x-dir. at y = ycell
 $var  nzby1 =         1      $end no. boundary particles in z-dir. at y = ycell
 $var  nxbz0 =         0      $end no. boundarz particles in x-dir. at z = zero
 $var  nybz0 =         0      $end no. boundarz particles in y-dir. at z = zero
 $var  nxbz1 =         0      $end no. boundarz particles in x-dir. at z = zcell
 $var  nybz1 =         0      $end no. boundarz particles in y-dir. at z = zcell
 $var  nfix =          0      $end no. of fixed particles
 $var  nfxpl =         2      $end no. of fixed planes
 $var  nzcyl =         0      $end no. z-cylinders
 $var  izeta =         0      $end flag for holding kin energy constant
 $var  icoord =        0      $end flag for coordinates print out
 $var  itty =          0      $end flag for tty interaction
 $var  ihertz =        0      $end flag for hertz power law
 $var  ixyz =          0      $end flag to read initial coords of fixed &
boundary particles
 $var ialtk =          0      $end flag for alternate values for side walls
 $var  tmax =     1.          $end max time for calculation
 $var  dt =       0.          $end time step
 $var  dtout =    0.          $end time interval for printing out results
 $var  dtoutx =   0.1         $end time interval for printing positions
 $var  dtoutv =   0.5         $end time interval for printing velocities
 $var  dtdump =   0.1         $end time interval for dumping
 $var  tzero =    1.          $end restart long-term cum. ave.
 $var  edot =     0.          $end strain rate
 $var  epsv =     0.          $end relative error tolerance
 $var  search = 0.005         $end search distance for near neighbors
 $var  ycell =  1.16250       $end cell height (m)
```

```
 $var  xyrat =  1.33333     $end ratio for xcell = 1.55
 $var  zyrat =  0.02        $end ratio for zcell = .0465/2
 $var  pack =    0.         $end solids packing fraction
 $var  vave =    0.0        $end average velocity (initial)
 $var  vseed =   0.5        $end seed for random initial particle velocities
 $var  vxzero =  0.0        $end initial velocity in the x-direction (ave)
 $var  vyzero =  0.0        $end initial velocity in y-direction (ave)
 $var  sknl =    2000.      $end normal force coefficient
 $var  elast =   0.85       $end coefficient of restitution
 $var  elastb =  0.85       $end  "
 $var  slope =   0.         $end alternative parameter for unloading
 $var  ratk =    0.7        $end ratio of tangential/normal stiffness
 $var  fmu =     0.5        $end coefficient of friction
 $var  fmub =    0.65       $end friction for boundary and fixed particles
 $var  drag =    0.0        $end v-squared drag force coefficient
 $var  power =   1.0        $end tangential force exponent
 $var  rmassz = 5032.3019   $end mass of unit sphere (makes rmass=5.92045e-4)
 $var  gravx =   9.80665    $end acceleration of gravity in y direction
 $var  draddt =  10.        $end rate of increase of particle radii
 $var  number(1) =      7000, 4          $end number of particles in each
group
 $var  radius(1) =     0.0049, 0.0049     $end particle radii each group
 $var  pmass(1) = 5.92045e-4, 5.92045e-4 $end particle mass in group
 $var  x(7003) = .2132117818, .7132117818 $end x location of fixed planes
 $var  y(7003) = .3463731207, .3463731207 $end y location of fixed planes
 $var  z(7003) =  .5,          .5         $end z location of fixed planes
 $var  q1old(7003) =  .0           ,  .0            $end fixed plane director
 $var  q2old(7003) =  .0           ,  .0            $end fixed plane director
 $var  q3old(7003) = -.461748613235, -.461748613235 $end fixed plane director
 $var  q4old(7003) =  .887010833178,  .887010833178 $end fixed plane director
 $var  finis =   1.         $end
```

Corrida 2. Diámetro máximo permisible para las partículas en el silo

```
i3ds002 35deg silo e=0.85, k=2000, fmu=0.5, fmub=0.65
 $var  np =          7004    $end total number of particles in cell
 $var  nfix =           0    $end number of fixed particles (lower boundary on
plane)
 $var  nout =           0    $end number of times to print out results
 $var  noutv =          0    $end number of times to print out velocities
 $var  nczero =         0    $end number of collisions before start cum.  ave.
 $var  ncmax =          0    $end number of collisions during entire run
 $var  ntcol =         40    $end number of timesteps during a collision.
 $var  nyzone =         1    $end number of y zones
 $var  nvel  =          1    $end number of velocity intervals
 $var  itervm =         1    $end max iterations per time step
 $var  ireal =          1    $end flag for periodic boundaries
 $var  imirr =          1    $end flag for mirror boundaries
 $var  nxby0 =          1    $end no. boundary particles in x-dir. at y = zero
 $var  nzby0 =          1    $end no. boundary particles in z-dir. at y = zero
 $var  nxby1 =          1    $end no. boundary particles in x-dir. at y = ycell
 $var  nzby1 =          1    $end no. boundary particles in z-dir. at y = ycell
 $var  nxbz0 =          0    $end no. boundarz particles in x-dir. at z = zero
 $var  nybz0 =          0    $end no. boundarz particles in y-dir. at z = zero
 $var  nxbz1 =          0    $end no. boundarz particles in x-dir. at z = zcell
 $var  nybz1 =          0    $end no. boundarz particles in y-dir. at z = zcell
 $var  nfix =           0    $end no. of fixed particles
 $var  nfxpl =          2    $end no. of fixed planes
 $var  nzcyl =          0    $end no. z-cylinders
 $var  izeta =          0    $end flag for holding kin energy constant
```

```
 $var  icoord =       0      $end flag for coordinates print out
 $var  itty =         0      $end flag for tty interaction
 $var  ihertz =       0      $end flag for hertz power law
 $var  ixyz =         0      $end flag to read initial coords of fixed &
boundary particles
 $var ialtk =         0      $end flag for alternate values for side walls
 $var  tmax =     1.         $end max time for calculation
 $var  dt =       0.         $end time step
 $var  dtout =    0.         $end time interval for printing out results
 $var  dtoutx =   0.1        $end time interval for printing positions
 $var  dtoutv =   0.5        $end time interval for printing velocities
 $var  dtdump =   0.1        $end time interval for dumping
 $var  tzero =    1.         $end restart long-term cum. ave.
 $var  edot =     0.         $end strain rate
 $var  epsv =     0.         $end relative error tolerance
 $var  search =  0.01        $end search distance for near neighbors
 $var  ycell =   1.16250     $end cell height (m)
 $var  xyrat =   1.33333     $end ratio for xcell = 1.55
 $var  zyrat =   0.04        $end ratio for zcell = .0465
 $var  pack =     0.         $end solids packing fraction
 $var  vave =     0.0        $end average velocity (initial)
 $var  vseed =    0.5        $end seed for random initial particle velocities
 $var  vxzero =   0.0        $end initial velocity in the x-direction (ave)
 $var  vyzero =   0.0        $end initial velocity in y-direction (ave)
 $var  sknl =     2000.      $end normal force coefficient
 $var  elast =    0.85       $end coefficient of restitution
 $var  elastb =   0.85       $end  "
 $var  slope =    0.         $end alternative parameter for unloading
 $var  ratk =     0.7        $end ratio of tangential/normal stiffness
 $var  fmu =      0.5        $end coefficient of friction
 $var  fmub =     0.65       $end friction for boundary and fixed particles
 $var  drag =     0.0        $end v-squared drag force coefficient
 $var  power =    1.0        $end tangential force exponent
 $var  rmassz =  5032.3019   $end mass of unit sphere (makes rmass=5.0323e-3)
 $var  gravx =     9.80665   $end acceleration of gravity in y direction
 $var  draddt =   10.        $end rate of increase of particle radii
 $var  number(1) =     7000, 4          $end number of particles in each group
 $var  radius(1) =     0.01, 0.01       $end particle radii each group
 $var  x(7003) = .2132117818, .7132117818 $end x location of fixed planes
 $var  y(7003) = .3463731207, .3463731207 $end y location of fixed planes
 $var  z(7003) =   .5,              .5          $end z location of fixed planes
 $var  q1old(7003) =  .0           ,  .0            $end fixed plane director
 $var  q2old(7003) =  .0           ,  .0            $end fixed plane director
 $var  q3old(7003) = -.461748613235, -.461748613235 $end fixed plane director
 $var  q4old(7003) =  .887010833178,  .887010833178 $end fixed plane director
 $var  finis =    1.         $end
```

Corrida 3. Estado permanente del flujo en un silo hexagonal

```
i3ds003 35deg silo e=0.85, k=2000, fmu=0.5, fmub=0.65
 $var  np =         12004    $end total number of particles in cell
 $var  nfix =           0    $end number of fixed particles (lower boundary on
plane)
 $var  nout =           0    $end number of times to print out results
 $var  noutv =          0    $end number of times to print out velocities
 $var  nczero =         0    $end number of collisions before start cum.  ave.
 $var  ncmax =          0    $end number of collisions during entire run
 $var  ntcol =         40    $end number of timesteps during a collision.
 $var  nyzone =         1    $end number of y zones
 $var  nvel  =          1    $end number of velocity intervals
```

```
 $var  istart =   1000   $end restart run
 $var  itervm =      1     $end max iterations per time step
 $var  ireal =       1     $end flag for periodic boundaries
 $var  imirr =       1     $end flag for mirror boundaries
 $var  nxby0 =       1     $end no. boundary particles in x-dir. at y = zero
 $var  nzby0 =       1     $end no. boundary particles in z-dir. at y = zero
 $var  nxby1 =       1     $end no. boundary particles in x-dir. at y = ycell
 $var  nzby1 =       1     $end no. boundary particles in z-dir. at y = ycell
 $var  nxbz0 =       0     $end no. boundarz particles in x-dir. at z = zero
 $var  nybz0 =       0     $end no. boundarz particles in y-dir. at z = zero
 $var  nxbz1 =       0     $end no. boundarz particles in x-dir. at z = zcell
 $var  nybz1 =       0     $end no. boundarz particles in y-dir. at z = zcell
 $var  nfix =        0     $end no. of fixed particles
 $var  nfxpl =       2     $end no. of fixed planes
 $var  nzcyl =       0     $end no. z-cylinders
 $var  izeta =       0     $end flag for holding kin energy constant
 $var  icoord =      0     $end flag for coordinates print out
 $var  itty =        0     $end flag for tty interaction
 $var  ihertz =      0     $end flag for hertz power law
 $var  ixyz =        0     $end flag to read initial coords of fixed &
boundary particles
 $var ialtk =        0     $end flag for alternate values for side walls
 $var  tmax =     4.       $end max time for calculation
 $var  dt =       0.       $end time step
 $var  dtout =    0.       $end time interval for printing out results
 $var  dtoutx =   0.5      $end time interval for printing positions
 $var  dtoutv =   0.5      $end time interval for printing velocities
 $var  dtdump =   0.2      $end time interval for dumping
 $var  tzero =    1.       $end restart long-term cum. ave.
 $var  edot =     0.       $end strain rate
 $var  epsv =     0.       $end relative error tolerance
 $var  search =  0.01      $end search distance for near neighbors
 $var  ycell =  1.16250    $end cell height (m)
 $var  xyrat =  1.33333    $end ratio for xcell = 1.55
 $var  zyrat =  0.1        $end ratio for zcell = .11625
 $var  pack =     0.       $end solids packing fraction
 $var  vave =     0.0      $end average velocity (initial)
 $var  vseed =    0.5      $end seed for random initial particle velocities
 $var  vxzero =   0.0      $end initial velocity in the x-direction (ave)
 $var  vyzero =   0.0      $end initial velocity in y-direction (ave)
 $var  sknl =     2000.    $end normal force coefficient
 $var  elast =    0.85     $end coefficient of restitution
 $var  elastb =   0.85     $end  "
 $var  slope =    0.       $end alternative parameter for unloading
 $var  ratk =     0.7      $end ratio of tangential/normal stiffness
 $var  fmu =      0.5      $end coefficient of friction
 $var  fmub =     0.65     $end friction for boundary and fixed particles
 $var  drag =     0.0      $end v-squared drag force coefficient
 $var  power =    1.0      $end tangential force exponent
 $var  rmassz = 5032.3019  $end mass of unit sphere (makes rmass=5.0323e-3)
 $var  gravx =    9.80665  $end acceleration of gravity in y direction
 $var  draddt =   10.      $end rate of increase of particle radii
 $var  number(1) =   12000, 4        $end number of particles in each group
 $var  radius(1) =     0.01, 0.01    $end particle radii each group
 $var  x(12003) = .2132117818, .7132117818 $end x location of fixed planes
 $var  y(12003) = .3463731207, .3463731207 $end y location of fixed planes
 $var  z(12003) =  .5,          .5         $end z location of fixed planes
 $var  q1old(12003) =  .0          ,  .0          $end fixed plane director
 $var  q2old(12003) =  .0          ,  .0          $end fixed plane director
 $var  q3old(12003) = -.461748613235, -.461748613235 $end fixed plane director
 $var  q4old(12003) =  .887010833178,  .887010833178 $end fixed plane director
```

```
 $var  finis =   1.          $end
```

Corrida 4. El silo completo

```
i3ds004 35deg silo e=0.85,  k=2000, fmu=0.5, fmub=0.65
 $var  np =        25006     $end total number of particles in cell
 $var  nfix =          0     $end number of fixed particles (lower boundary on
plane)
 $var  nout =          0     $end number of times to print out results
 $var  noutv =         0     $end number of times to print out velocities
 $var  nczero =        0     $end number of collisions before start cum.  ave.
 $var  ncmax =         0     $end number of collisions during entire run
 $var  ntcol =        40     $end number of timesteps during a collision.
 $var  nyzone =        1     $end number of y zones
 $var  nvel  =         1     $end number of velocity intervals
 $var  istart =        0   $end restart run
 $var  itervm =        1     $end max iterations per time step
 $var  ireal =         1     $end flag for periodic boundaries
 $var  imirr =         1     $end flag for mirror boundaries
 $var  nxby0 =         1     $end no. boundary particles in x-dir. at y = zero
 $var  nzby0 =         1     $end no. boundary particles in z-dir. at y = zero
 $var  nxby1 =         1     $end no. boundary particles in x-dir. at y = ycell
 $var  nzby1 =         1     $end no. boundary particles in z-dir. at y = ycell
 $var  nxbz0 =         0     $end no. boundarz particles in x-dir. at z = zero
 $var  nybz0 =         0     $end no. boundarz particles in y-dir. at z = zero
 $var  nxbz1 =         0     $end no. boundarz particles in x-dir. at z = zcell
 $var  nybz1 =         0     $end no. boundarz particles in y-dir. at z = zcell
 $var  nfix =          0     $end no. of fixed particles
 $var  nfxpl =         3     $end no. of fixed planes
 $var  nzcyl =         0     $end no. z-cylinders
 $var  izeta =         0     $end flag for holding kin energy constant
 $var  icoord =        1     $end flag for coordinates print out
 $var  itty =          0     $end flag for tty interaction
 $var  ihertz =        0     $end flag for hertz power law
 $var  ixyz =          0     $end flag to read initial coords of fixed &
boundary particles
 $var ialtk =          0     $end flag for alternate values for side walls
 $var  tmax =     3.         $end max time for calculation
 $var  dt =       0.         $end time step
 $var  dtout =    0.         $end time interval for printing out results
 $var  dtoutx =   0.2        $end time interval for printing positions
 $var  dtoutv =   0.2        $end time interval for printing velocities
 $var  dtdump =   0.05       $end time interval for dumping
 $var  tzero =    1.         $end restart long-term cum. ave.
 $var  edot =     0.         $end strain rate
 $var  epsv =     0.         $end relative error tolerance
 $var  search =  0.00101     $end search distance for near neighbors
 $var  ycell =   .11625      $end cell height (m)
 $var  xyrat =  1.72         $end ratio for xcell = .2
 $var  zyrat =  0.1098       $end ratio for zcell = .012767
 $var  pack =     0.         $end solids packing fraction
 $var  vave =     0.0        $end average velocity (initial)
 $var  vseed =    0.5        $end seed for random initial particle velocities
 $var  vxzero =   0.0        $end initial velocity in the x-direction (ave)
 $var  vyzero =   0.0        $end initial velocity in y-direction (ave)
 $var  sknl =     2000.      $end normal force coefficient
 $var  elast =    0.85       $end coefficient of restitution
 $var  elastb =   0.85       $end  "
 $var  slope =    0.         $end alternative parameter for unloading
 $var  ratk =     0.7        $end ratio of tangential/normal stiffness
```

```
 $var  fmu =      0.5        $end coefficient of friction
 $var  fmub =     0.65       $end friction for boundary and fixed particles
 $var  drag =     0.0        $end v-squared drag force coefficient
 $var  power =    1.0        $end tangential force exponent
 $var  rmassz = 5032.3019    $end mass of unit sphere (makes rmass=5.0323e-6)
 $var  gravx =     9.80665   $end acceleration of gravity in y direction
 $var  draddt =   10.         $end rate of increase of particle radii
 $var  number(1) = 25000, 2,    2,    1,    1 $end number of particles in each
group
 $var  radius(1) = .001, .001, .001, .001, .001 $end particle radii each group
 $var  planex(1) =  1.,  1.,  .316638, .215405, .363912$end x' length of fixed
planes
 $var  planez(1) =  1.,  1.,  1.,       1.,       .2$end z' length of fixed
planes
 $var  x(25003) = .126259, .818385, .538950, .257325
$end x location of fixed planes
 $var  y(25003) = .392902, .446237, .588905, .557289
$end y location of fixed planes
 $var  z(25003) = .5, .5, .5, .5
$end z location of fixed planes
 $var  q1old(25003) =  .0, .0,.0,
.0
$end fixed plane director
 $var  q2old(25003) =  .0, .0,.0,
.0
$end fixed plane director
 $var  q3old(25003) =   .548293229520, -.461748613235, -.461748613235,
.382683432365
$end fixed plane director
 $var  q4old(25003) =  .836286155848,  .887010833178,  .887010833178,
.923879532511
$end fixed plane director
 $var  finis =   1.         $end
```

Programas de Visualización

Para poder interpretar los resultados obtenidos de una simulación numérica se vuelve generalmente necesario generar algún tipo de imagen o gráfica. Esto se debe a que los números por sí solos, aún cuando sepamos lo que cada número representa, son difíciles de visualizar. Es necesario entonces emplear alguna clase de postprocesamiento que transforme los fríos números en algo un poco más comprensible.

Para la interpretación de las simulaciones de `3dshear` se empleó un par de programas igualmente en Fortran77 que, dados como entrada los archivos de resultados de `3dshear`, genera una serie de imágenes PostScript que corresponden a rebanadas del flujo a diferentes tiempos y con mayor o menor grado de perspectiva (para el caso de las posiciones).

El primero de estos programas, `plotmirr`, muestra las posiciones de las partículas representando a éstas como círculos. Este programa está escrito específicamente para geometrías con simetría, de donde toma su nombre de *dibuja espejo*.[a] Las partículas más lejanas se representan con tonos más obscuros, y las más cercanas, con tonos más claros. En caso de que se desee visualizar la orientación de cada partícula, el programa puede representar por medio de arcos elípticos el ecuador y el meridiano principal de cada partícula. La descripción detallada de cada uno de los parámetros de entrada se hace en el código mismo, que se enlista a continuación.

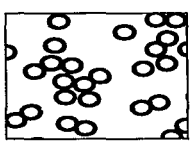
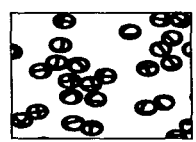


Figura 19. La figura a la izquierda muestra una configuración sin hacer énfasis en las orientaciones de cada partícula. La figura a la derecha, en cambio, muestra la misma configuración incluyendo las orientaciones.



[a] En inglés, *plot mirror*.

```
c     f77 +U77 -O -o xpltmirr pltmirr.f
c
c     This routine reads  configurations of x, y, z points from input
c     file 'config.plot' and then generates a postscript plot file of
c     the configurations.
c
c     Otis Walton, L-207, LLNL, P.O. Box 808, Livermore, CA 94550
c     (415)422-3947, FAX(415)422-3118, e-mail: walton@s85.es.llnl.gov
c
c     Adapted from /us/walton/plot3d/plotthin.f   5/28/91
c
c     10/27/95 modified to read o3dx or o3dxq files from 3dshear
c     (i.e., header line,
c           skip a line,
c           n1 and n2,
c           r1 and r1,
c           xcell, ycell and zcell,
c           time,
c           xx,yy,zz  (for n1+n2 lines)
c           time,
c           xx,yy,zz  (for n1+n2 lines)
c           etc.                        )
c
c
c     ------------ DESCRIPTION OF VARIABLES --------
c
c The parameter 'nmx' is the maximum number of spheres on a single frame
c
c       n1     = no. of spheres of size 1
c       n2     = no. of spheres of size 2
c       ns     = n1 + n2 = total number of spheres in each frame.
c       r1     = radius for first n1 spheres read in (i.e. group 1)
c       r2     = radius for next n2 spheres read in (i.e., group 2)
c       x, y, z = input coordinates of sphere centroids for configuration
c       xp, yp, zp = sphere coordinates, in order of increasing z
c       np     = particle number (original input order)
c       ndx    = np value of reordered spheres (i.e., particle number)
c       xcell = width of cell to be plotted
c       ycell = height of cell
c       zcell = depth of cell
c               Note, these are currently data loaded to 1.0
c       zh     = z-thickness of thin section of second image cell plotted
c
c       eta, edot, ce, fmu, beta, eps, and time are quantities used only
c       in the output label for each configuration.
c
c       variables alpha, zscale and zvanish control the shape of the
c       perspective projection of the unit cell
c
c       unit 5 = standard input --- assumed to be interactive keyboard
c       unit 6 = standard output --- assumed to be workstation monitor
c       unit 7 = input file 'config.plot', 3 lines header, then x,y,z
c       unit 8 = postscript output file, 'plotout.ps' (i.e., pictures!)
c                use 'lpr plotout.ps' to send postscript file to printer
c       unit 9 = output file 'debugedit' containing i/o messages & errors

c        Note: This routine was written to read a specific format output
c              file generated by the hard sphere code hsbmx.  Several of
c              input and output lines are specific to that application.
c
c     Configurations are read, one at a time into the arrays x, y, & z,
```

```
c      then they are reordered in order of increasing z value so that
c      closer spheres will 'cover' spheres further away in the final
c      post script picture.  The reordered coordinated xp, yp, zp are
c      given to subroutine 'plot' where they are scaled appropriately to
c      make a perspective-like image.
c
c      The arrays z and np are changed during the reordering and do not
c      contain useful information after reordering is complete.
c ---------------------------------------------------------------------

       implicit real*8(a-h,o-z)
       real*4 alpha1,zscale1,zvan1
       real etime, tarray(2)
       character line1, label, idum
       parameter (NMX = 60000)
       dimension x(nmx),y(nmx),z(nmx),np(nmx)
      >         ,xp(nmx),yp(nmx),zp(nmx),ndx(nmx)
      >         ,rotx(nmx),roty(nmx),rotz(nmx)
      >         ,rotxp(nmx),rotyp(nmx),rotzp(nmx)
       dimension idum(10)
       common/ispheres/ ns,ns2,n1,n2,iellipse
       common/spheres/ r1,r2,ce,eta,fmu,beta
       common/cell/xcell,ycell,zcell,alpha,zscale,zvanish,zh,edot,time
       common/label/line1(80)
c
       data alpha/45./,zscale/0.4/,zvanish/2.0/,xcell/1./,iellipse/0/
       data ycell/1./,zcell/1./,zh/1.0/
       write(6,*)' alpha,zscale,zvanish,iellipse  (zero for defaults): ',
      >          '45., 0.4, 2., 0'
       read(5,*) alpha1,zscale1,zvan1,iel1
       if(alpha1.ne.0.) alpha = alpha1
       if(zscale1.ne.0.) zscale = zscale1
       if(zvan1.ne.0.) zvanish = zvan1
       if(iel1.ge.0) iellipse = iel1
       write(6,fmt="(' using:',3f6.2,i3)")alpha,zscale,zvanish,iellipse
       write(6,*)' deltaz (zero for default):',
      >          '1.0'
       read(5,*) deltaz
       if(deltaz.ne.0.) zh = deltaz
       open(unit=9, file="debugedit", form="formatted")
       write(9,*)' alpha,zscale,zvanish,iellipse:',
      >          alpha,zscale,zvanish,iellipse

       open(unit=7,file="config.plot",form="formated",status="old",
      >     err=900)
       nconf=0
       read(7,fmt="(80a1)",err=900)(line1(i),i=1,80)
c-------------------------skip line (3dshear ... version...)
       read(7,fmt="(10a8)",err=910)(idum(i),i=1,10)
       read(7,*,err=920)label,n1,n2
       read(7,*,err=930)label,r1,r2
       read(7,*,err=980)label,xcell,ycell,zcell
       yref = 2.*(ycell - r2)
       zh = zh*zcell
       zvanish = zvanish*zcell
       ns = n1 + n2
       ns2 = ns + ns
       write(6,fmt="('xcell,ycell,zcell,n1,n2,r1,r2:',3e10.3,2i5,3e12.4)"
      >)xcell,ycell,zcell,n1,n2,r1,r2
       write(9,fmt="('xcell,ycell,zcell,n1,n2,r1,r2:',3e10.3,2i5,3e12.4)"
      >)xcell,ycell,zcell,n1,n2,r1,r2
```

```
  100   read(7,*,err=985,end=985)label,time
        write(6,fmt="('time:',0pf10.3,3x,'ns:',
     >              i5)")time,ns
        write(9,fmt="('time:',0pf10.3,3x,'ns:',
     >              i5)")time,ns


        do 115 i=1,ns
        read(7,*,err=970) xx,yy,zz,rotxx,rotyy,rotzz
        imod = i/2
        iodd = i - imod*2
        x(2*i - 1) = xx
        z(2*i - 1) = zz
        roty(2*i - 1) = rotyy
        x(2*i) = xx
        z(2*i) = zz
        roty(2*i) = rotyy
        if(iodd.eq.1) then
 y(2*i - 1) = yy
 y(2*i) = yref - yy
 rotx(2*i - 1) = rotxx
 rotx(2*i) = -rotxx
 rotz(2*i - 1) = rotzz
 rotz(2*i) = -rotzz
        else
           y(2*i - 1) = yref - yy
           y(2*i) = yy
           rotx(2*i - 1) = -rotxx
           rotx(2*i) = rotxx
           rotz(2*i - 1) = -rotzz
           rotz(2*i) = rotzz
        endif
        np(2*i - 1) = 2*i - 1
        np(2*i) = 2*i
   115  continue

        nconf=nconf + 1
c
c--------reorder position vectors in increasing value of z
        big = 1.e+20
        k=0
        nlist = ns2
   400  zmin1 = big
c-----find smallest remaining z value
        do 500 i = 1,nlist
  if(z(i).gt.zmin1) go to 500
  zmin1 = z(i)
  imin = i
   500  continue
c
c------add smallest remaining z to end of reordered list
        k = k + 1
        ndx(k) = np(imin)
        zp(k) = z(imin)
        xp(k) = x(ndx(k))
        yp(k) = y(ndx(k))
        rotxp(k) = rotx(ndx(k))
        rotyp(k) = roty(ndx(k))
        rotzp(k) = rotz(ndx(k))
        if(k.ge.ns2) go to 600
        nlist = nlist - 1
```

```
      if(nlist.lt.imin) go to 400
c
c-----move remaining z and np values up.  note else remain in orig order
      do 550 i = imin,nlist
 z(i) = z(i+1)
 np(i) = np(i+1)
  550 continue
      go to 400
c ----done with reorder on z
  600 continue

      call plot(xp,yp,zp,rotxp,rotyp,rotzp,ndx)

c
c     continue reading configurations and plotting until file exhausted
c
      go to 100

c
  900 write(6,*)' could not open file config.plot'
      write(9,*)' could not open file config.plot'

      go to 999
  910 write(6,*)' error reading 2nd line of config.plot'
      write(9,*)' error reading 2nd line of config.plot'
      go to 990

  920 write(6,*)' error reading number of spheres, n1 and n2'
      write(9,*)' error reading number of spheres, n1 and n2'
      go to 990

  930 write(6,*)' error reading radii, r1 and r1'
      write(9,*)' error reading radii, r1 and r1'
      go to 990

  970 ii= i-1
      write(6,9970)ns,ii, x(ii), y(ii), z(ii)
      write(9,9970)ns,ii, x(ii), y(ii), z(ii)

 9970 format(' reading',i5,' records, last read i=',i5,' x,y,z:',
     >       1p3e12.4)
      go to 990
  980 write(6,*)' error reading xcell, ycell, zcell file config.plot'
      write(9,*)' error reading xcell, ycell, zcell file config.plot'

      go to 990
  985 write(6,fmt="(' error reading time, last time read:',
     >                1p2e13.6)")time
      write(9,fmt="(' error reading time, last time read:',
     >                1p2e13.6)")time

  990 write(6,fmt="(' processed',i5,' configurations')") nconf
      write(9,fmt="(' processed',i5,' configurations')") nconf
c-----computer time used
  999 ttot = etime(tarray)
      ttot = tarray(1) + tarray(2)
      write(6,fmt="(' cumulative computer time (s) =', 1pe12.4)")ttot
      write(9,fmt="(' cumulative computer time (s) =', 1pe12.4)")ttot
      stop
      end
c
```

```
c
c
      subroutine plot(x,y,z,xrot,yrot,zrot,np)
c     Plots sphere positions by generating a postscript file.
c-----
c     This routine generates a crude one vanishing point perspective
c     projection of sections of a cell.
c     Particles are represented as filled circles (with ellipses for
c     equator and prime meridian lines on each sphere).
c
c                        3_____4          3_____4
c                      13/____14/|         |     |
c                        |  o  ||         |     |
c                        | o  o||         |_____|
c                        |_____/2        1     2
c                       11     12
c
c     Image characteristics are controlled by four parameters:
c      alpha: the angle the projected z-axis makes with the x-axis
c      zscale:  zunit/xunit, unit of measure along projected z-axis
c               (uniformly applied)
c      zvanish - distance along z-axis (in x-axis units) to where all
c               z-lines converge.
c      zh:- the thickness of the sections thru the cell along the z-axis
c
c     This scaling could, alternatively, be accomplished by
c     specifying the x and y coordinates of the z-vanishing point and
c     an appropriate scale to apply at the origin, with the scale
c     varying as the inverse of the distance from the vanishing pt.
c     This would be only slightly more 'correct' than the current
c     method since x and y axes would still be horizontal and vertical.
c ---ADDED---(7/18/96)
c     Besides, the code does not adjust the rotations of the ellipses
c     to show the different angles in the perspective.
c -----------
c
      implicit real*8(a-h,o-z)
      real*4 graz
      character line1
      parameter (NMX = 60000, MXSLICE=10)
      dimension x(nmx),y(nmx),z(nmx),np(nmx),nsl(mxslice)
     >         ,xrot(nmx),yrot(nmx),zrot(nmx)
      common/ispheres/ ns,ns2,n1,n2,iellipse
      common/spheres/ r1,r2,ce,eta,fmu,beta
      common/cell/xcell,ycell,zcell,alpha,zscale,zvanish,zh,edot,time
      common/label/line1(80)

      dimension mp(2,14)
      dimension pt(3,14),pp(2,14)

c       zh =   thickness of each section to be plotted (user coords)
c       pt(ijk,m) ----alternate variable for same 14 cornerpoints
c       pp(ij,m) -----two dimensional projection of 14 corner points
c       mp(ij,m) -----post script integer coordinates of 14 pts
c-------------------------------------------------------------------
c
c----- initailize positions of corners
c
      data npage/0/

      do 10 m=1,4
```

```
   10  pt(3,m)= 0.
       do 11 m=11,14
   11  pt(3,m)= zh

       pt(2,1)= 0.
       pt(2,2)= 0.
       pt(2,11)= 0.
       pt(2,12)= 0.
       pt(2,3)= ycell
       pt(2,4)= ycell
       pt(2,13)= ycell
       pt(2,14)= ycell

       pt(1,1)= 0.
       pt(1,11)= 0.
       pt(1,3)= 0.
       pt(1,13)= 0.
       pt(1,2)= xcell
       pt(1,4)= xcell
       pt(1,12)= xcell
       pt(1,14)= xcell

      open(unit=8, file='plotout.ps')

c programming note (9/19/90):
c                 Prologue info is repeated for each config in output
c                 file.  This is not necessary.  Eliminating this dup-
c                 lication would slightly reduce size of output file.
c                 Also, open statement is encountered each time thru
c                 (again, not necessary, but it doesn't seem to cause
c                 any problems).

      write(8,fmt="('%!PS-Adobe-1.0')")
      write(8,fmt="('% lib/pscat.pro -- prolog for pscat (troff) files
     >')")
      write(8,fmt="('% Copyright (C) 1985 Adobe Systems, Inc.')")
      write(8,*) 'save /pscatsave exch def'
      write(8,*) '/$pscat 50 dict def'
      write(8,*) '$pscat begin'
      write(8,fmt="('%%EndProlog')")
c
      zero = 0.0
      one = 1.0
      two = 2.
      three = 3.
      four = 4.
      pi = four*atan(one)
      degrad = pi/180.
      alphar = alpha*degrad
      cosa = cos(alphar)
      sina = sin(alphar)
      width = 2000*xcell/ycell
      height = 2000
      xoffset = zscale*cosa*zh
      yoffset = zscale*sina*zh
      factor = width/xcell
      nx0 = 90 + xoffset*factor
      ny0 = 100 + yoffset*factor
c
      zvi = one/zvanish
```

```
c----scale coordinates
      do 100 m=1,14
        zp = zscale*pt(3,m)
        xyscale = one + zvi*zp
        pp(1,m) = pt(1,m)*xyscale - zp*cosa
        pp(2,m) = pt(2,m)*xyscale - zp*sina
        mp(1,m) = factor*pp(1,m)
mp(2,m) = factor*pp(2,m)
  100 continue

      if(iellipse.ne.0) then

c     *****define ellipses*********
c       Copied from PostScript Cookbook by Adobe Systems, (c)1985
c                    Addison-Wesley Pub., 1987
        write(8,*) ' /ellipsedict 8 dict def'
        write(8,*) ' ellipsedict /mtrx matrix put'
        write(8,*) ' /ellipse'
        write(8,*) ' { ellipsedict begin'
        write(8,*) '   /endangle exch def'
        write(8,*) '   /startangle exch def'
        write(8,*) '   /yrad exch def'
        write(8,*) '   /xrad exch def'
        write(8,*) '   /y exch def'
        write(8,*) '   /x exch def'

        write(8,*) '   /savematrix mtrx currentmatrix def'
        write(8,*) '   x y translate'
        write(8,*) '   xrad yrad scale'
        write(8,*) '   0 0 1 startangle endangle arc'
        write(8,*) '   savematrix setmatrix'
        write(8,*) ' end'
        write(8,*) ' } def'
c     ******end def ellipses ************
      endif


c..... select subset of spheres in each slice
       nframes = zcell/zh
      if(nframes.gt.mxslice) nframes = mxslice

islice = 1
        zmax = islice*zh

      do 120 j = 1,ns2
        if(z(j).le.zmax) then
          nsl(islice) = j
  go to 120
else
  if(islice.ge.mxslice) go to 121
  islice = islice + 1
          nsl(islice) = j
          zmax = islice*zh
endif
  120 continue
  121 continue
      nslice = islice

c...  loop over slices drawing a box around each slice
      do 300 is=1,nslice
```

```
        zmin = (is-1)*zh
        zmax = is*zh

      npage = npage + 1
      write(8,fmt="('%%Page:',2i5)")npage,npage
      write(8,*) 'initgraphics 72 300 div dup scale newpath'
      write(8,*) '/normalfont'
      write(8,*) '/Helvetica-Bold findfont 60 scalefont def'
      write(8,*) 'normalfont setfont'
      write(8,*) '3 setlinewidth'
      write(8,*) '0 200 translate % move away from lower left corner'
      write(8,*) '.9 .9 scale'
c***** write(8,*)' 1800 100 translate % move to rotation point'
c***** write(8,*)' 90 rotate            % landscape orientation'
      write(8,*)nx0,ny0,' translate % origin at rear left corner (pt1)'

c      write(8,*)mp(1,1),mp(2,1),' moveto ',mp(1,2),mp(2,2),' lineto ',
c     >           mp(1,4),mp(2,4),' lineto'
c      write(8,*)mp(1,3),mp(2,3), ' lineto ', mp(1,1),mp(2,1),
c     >           ' lineto closepath stroke'
c      write(8,*)mp(1,1),mp(2,1),' moveto ',mp(1,11),mp(2,11),
c     >           ' lineto closepath stroke'
c      write(8,*)mp(1,2),mp(2,2),' moveto ',mp(1,12),mp(2,12),
c     >           ' lineto closepath stroke'
c      write(8,*)mp(1,3),mp(2,3),' moveto ',mp(1,13),mp(2,13),
c     >           ' lineto closepath stroke'
c      write(8,*)mp(1,4),mp(2,4),' moveto ',mp(1,14),mp(2,14),
c     >           ' lineto closepath stroke'


      ibox2 = 0
      nfirst = 1
      if(is.ne.1) nfirst = nsl(is -1) + 1
      nlast = nsl(is)
      do 200 i=nfirst,nlast
rad = r1
if(np(i).gt.n1)rad = r2
c..........shift particles in each slice to location of first for plot

        zp = zscale*(z(i) - zmin)
        xyscale = one + zvi*zp
        xx = x(i)*xyscale - zp*cosa
        yy = y(i)*xyscale - zp*sina
        nx = xx*factor
        ny = yy*factor
znear = zh - rad
graz = 0.5 + 0.5*(z(i)-zmin)/znear

scaler = rad*xyscale
nrad = scaler*factor

c     finish drawing the front of primary bounding box before plotting
c     front spheres.
c      if(((z(i) - zmin).gt.znear).and.(ibox2.eq.0)) then
c ibox2 = 1
c         write(8,*)mp(1,11),mp(2,11),' moveto ',mp(1,12),mp(2,12),
c     >           ' lineto',mp(1,14),mp(2,14),' lineto '
c         write(8,*)mp(1,13),mp(2,13), ' lineto ', mp(1,11),mp(2,11),
c     >           ' lineto closepath stroke'
c         write(8,*)mp(1,6),mp(2,6),' moveto ',mp(1,10),mp(2,10),
c     >           ' lineto closepath stroke'
```

```
c       endif
c
if(iellipse.ne.0) then
c---     calculate ellipse properties
c---     new code for arbitrary rotations about +x +y +z
            sinxrot = sin(xrot(i))
            sinyrot = sin(yrot(i))
            prodsin = sinxrot*sinyrot
            xyrotr = -asin(sinxrot*sinyrot)
            xyrotd = xyrotr/degrad
            zrotd = zrot(i)/degrad
            nxrad = scaler*factor*sinyrot
            nyrad = scaler*factor*sinxrot*cos(yrot(i))
            if(prodsin.lt.0) nrad = -nrad
c----------Commented out old code----------------
c           zf = (z(i) - zmin) + rad
c           zb = (z(i) - zmin) - rad
c           zpf = zscale*zf
c           zpb = zscale*zb
c           xysf = one + zvi*zpf
c           xysb = one + zvi*zpb
c           xxf = x(i)*xysf - zpf*cosa
c           yyf = y(i)*xysf - zpf*sina
c           xxb = x(i)*xysb - zpb*cosa
c           yyb = y(i)*xysb - zpb*sina
c           dy = yyb - yyf
c           dx = xxb - xxf
c           absdy = abs(dy)
c           absdx = abs(dx)
c           absry = absdy/two
c           absrx = absdx/two
c           nxrad = absrx*factor
c           nyrad = absry*factor
c           arcx = ' 180 360'
c           arcy = ' 90 270 '
c           if(dx.lt.zero) arcy = ' 270 90 '
c           if(dy.lt.zero) arcx = '  0 180 '
c-----------------------------------------------
         endif
c
c---    plot scaled circle ... filled
         write(8,*) nx,ny,nrad,' 0 360 arc closepath '
c------    note:
c               Preliminary version plotted transparent spheres near
c               front with 360 degree ellipses.  Resulting view was
c               confusing and difficult to interpret.
c
   if(graz.gt.1.)graz = 1.
      write(8,fmt="(' gsave ',f7.3,' setgray fill grestore stroke ')")
     >          graz
   if(iellipse.ne.0) then
c---           plot ellipses in x-z and x-y planes on circle
               write(8,*)'newpath gsave'
               write(8,*) nx,ny,' translate'
               write(8,fmt="(f7.2,' rotate')")zrotd
               write(8,*)' 0 0 ',nxrad,nrad,' 270 90 ellipse  stroke'
               write(8,fmt="(f7.2,' rotate')")xyrotd
               write(8,*)' 0 0 ',nrad,nyrad,' 180 360 ellipse  stroke'
               write(8,*)'grestore'
   endif
```

```
 200    continue
 c
 c      finsih front of box if not already done
 c       if(ibox2.eq.0) then
 c          write(8,*)mp(1,11),mp(2,11),' moveto ',mp(1,12),mp(2,12),
 c      >            ' lineto',mp(1,14),mp(2,14),' lineto '
 c          write(8,*)mp(1,13),mp(2,13), ' lineto ', mp(1,11),mp(2,11),
 c      >            ' lineto closepath stroke'
 c          write(8,*)mp(1,6),mp(2,6),' moveto ',mp(1,10),mp(2,10),
 c      >            ' lineto closepath stroke'
 c       endif

 c---- write title info above plot of configuration
       write(8,*)' 0 340 moveto (',line1,') show'
       write(8,*)' 0 260 moveto '
       write(8,fmt="('( n1=',i6,'  n2=',i6,'   r1=',1pe12.4,
      > '    r2=',1pe12.4,') show')")n1,n2,r1,r2
       write(8,fmt="(' 0 130 moveto (Time =',f12.5,') show')") time

       nps = nlast - nfirst + 1
       write(8,fmt="(' 0 50 moveto (Slice with ',i6,' particles) show'
      > )") nps
       write(8,*) 'showpage'

   300 continue
 c
 c      (Trailer info written after each showpage.  Not necessary!)
 c
       write(8,fmt="('%%Trailer')")
       write(8,*) 'pscatsave end restore'
       write(8,fmt="('%%Pages: ',i3)")npage

 c----       Note: system call to lpr causes errors and lost output on
 c                 Solbourne/ Sun sparc system (unknown cause)
 c-----       close(8)
 c----        idummy = system('lpr plotout.ps')

       return
       end
```

El segundo programa de postprocesamiento empleado lleva el nombre de `plotvect`, pues lo que hace es graficar vectores.[a] El sistema es básicamente igual al que sigue `plotmirr`, pero se requiere de dos archivos de entrada en vez de uno: un archivo para la posición de las partículas y otro archivo para la velocidad de las mismas.

Se grafica un conjunto de vectores cuyas cola se encuentran exactamente en la posición correspondiente a cada partícula del sistema. La dirección de cada vector será igual a la dirección de la velocidad de su partícula correspondiente. La magnitud y el tono de gris de los vectores son proporcionales al logaritmo de la rapidez de la partícula representada. Se logra con esto obtener mapas de velocidades como el mostrado en la *Figura 17a*, pág. 44. A continuación se muestra la rutina empleada para dibujar las flechas.

```
c  ********* define arrows *********
c       Copied from PostScript Cookbook by Adobe Systems, (c)1985
c                 Addison-Wesley Pub., 1987
      write(8,*) '/arrowdict 10 dict def'
      write(8,*) 'arrowdict begin'
      write(8,*) '  /mtrx matrix def'
      write(8,*) 'end'
      write(8,*) '/arrow'
      write(8,*) '{ arrowdict begin'
      write(8,*) ' /headlength exch def'
      write(8,*) ' /halfheadthickness exch 2 div def'
      write(8,*) ' /halfthickness exch 2 div def'
      write(8,*) ' /angle exch def /arrowlength exch def'
      write(8,*) ' /taily exch def /tailx exch def'
      write(8,*) ' /base arrowlength headlength sub def'
      write(8,*) ' /savematrix mtrx currentmatrix def'
      write(8,*) ' tailx taily translate'
      write(8,*) ' angle rotate'
      write(8,*) ' 0 halfthickness neg moveto'
      write(8,*) ' base halfthickness neg lineto'
      write(8,*) ' base halfheadthickness neg lineto'
      write(8,*) ' arrowlength 0 lineto'
      write(8,*) ' base halfheadthickness lineto'
      write(8,*) ' base halfthickness lineto'
      write(8,*) ' 0 halfthickness lineto'
      write(8,*) ' closepath'
      write(8,*) ' savematrix setmatrix'
      write(8,*) ' end'
      write(8,*) '} def'
c  ********* end def arrows *********
```

[a] En inglés, *plot vectors*.

Referencias

Adobe Systems, 1987, *PostScript Cookbook*, Addison–Wesley.

Allen, M. P., y D. J. Tildesley, 1987, *Computer Simulation of Liquids*, Oxford Science Publications.

Aoki, K. M., T. Akiyama, Y. Maki, y T. Watanabe, 1996, Phys. Rev. E **54**, 874.

Ben-Naim, E., J. B. Knight, E. Nowak, H. M. Jaeger, y S. R. Nagel, 1996, preprint Phys. Rev. Lett.

Bideau, D., y J. A. Dodds, 1991, editores, *Physics of Granular Media*, Serie Les Houches, Nova Science.

Brooker, D. B., F. W. Bakker–Arkema, y C. W. Hall, 1992, *Drying and storage of grains and oilseeds*, Van Nostrand Reinhold, pp. 133–135 y apéndices.

Campbell, C. S., 1990, Annu. Rev. Fluid Mech. **22**, 57.

Chávez Montes, B. E., 1997, *Poscosecha del Maiz en México. Evaluación de una Alternativa: el Silo Solar Hexagonal*, Tesis de Licenciatura, Fac. de Química.

Coulomb, C., 1773, en *Memoir de Mathematique et de Physique*, Vol. 7, Academie des Sciences, Paris, p. 343.

Ehrichs, E. E., H. M. Jaeger, G. S. Karczmar, J. B. Knight, V. Yu. Kuperman, y S. R. Nagel, 1995, Science **267**, 1632.

Ennis, B. J., J. Green, y R. Davis, 1994, Particle Technology **90**, 32.

Evans, D. J., y S. Murad, 1977, Mol. Phys. **34**, 327.

Faraday, M., 1831, Phil. Trans. R. Soc. London **52**, 299.

Galicia Ávila, O. M., 1997, *Algunas Consideraciones Termodinámicas en la Conservación de Granos Alimenticios*, Tesis de Licenciatura, Fac. de Ingeniería.

Goldstein, H., 1950, *Classical Mechanics*, Addison–Wesley, p. 9, § 10.

Haff, P. K., 1983, J. Fluid Mech. **134**, 401.

Haff, P. K., 1986, J. Rheol. **30**, 931.

Hernández Nava, G., H. R. Morano Okuno, L. G. Sosa Lujano, 1995, *Secado y Aireación Solar en un Silo Hexagonal*, Tesis de Licenciatura, Fac. de Ingeniería.

Hewlett Packard, 1996, `HP-UX 10.01 man pages`.

Hill, K. M., A. Caprihan, y J. Kakalios, 1997, Phys. Rev. Lett. **78**, 50.

Ippolito, I., C. Annic, J. Lemaitre, L. Oger, y M. Matsushita, 1995, Phys. Rev. E **52**, 2072.

Jaeger, H. M., S. R. Nagel, y R. P. Behringer, 1996a, Phys. Today **49**, 32.

Jaeger, H. M., S. R. Nagel, y R. P. Behringer, 1996b, Rev. Mod. Phys. **68**, 1259.

Jenkins, J. T., y S. B. Savage, 1983, J. Fluid Mech. **130**, 186.

Joseph, G., B. Mena, E. Moreno, E. Sansores y O. R. Walton, 1996, *Proceedings XIIth International Congress on Rheology*, editado por A. Aït-Kadi, J. M. Dealy, D. F. James y M. C. Williams, Québec, Canada, 795.

Knight, J. B., C. G. Fandrich, C. N. Lau, H. M. Jaeger, y S. R. Nagel, 1995, Phys. Rev. E **51**, 3957.

Knight, J. B., H. M. Jaeger, y S. R. Nagel, 1993, Phys. Rev. Lett. **70**, 3728.

Knowlton, T. M., J. W. Carson, G. E. Klinzing, y W.-C. Yang, 1994, Particle Technology **90**, 44.

Lee, J., 1994a, J. Phys. A **27**, L257.

Lee, J., 1994b, Phys. Rev. E **49**, 281.

Liu, C. H., S. R. Nagel, D. A. Schecter, S. N. Coppersmith, S. Majumdar, O. Narayan, y T. A. Witten, 1995, Science **269**, 513.

McNamara, S., y Young, V. R., 1994, Phys. Rev. E **50**, 28.

Metcalfe, G., T. Shinbrot, J. J. McCarthy, y J. M. Ottino, 1995, Nature **374**, 39.

Mindlin, R. D., 1949, J. Appl. Mech. Trans. ASME E **16**, 259.

Morris, S. W., y K. Choo, 1996, http://mobydick.physics.utoronto.ca/sand.html (Mayo 9, 1997), Grupo Experimental de Física No–Lineal, Universidad de Toronto.

Nakagawa, M., y A. Caprihan, 1994, *Meeting on Flows of Granular Materials in Complex Geometries*, editado por S. L. Passman, E. Fukushima y R. E. Evans, Sandia Report **SAND94–2732·UC–117**, 19.

Ogawa, S., 1978, *Proceedings US-Japan Seminar on Continuum-Mechanical and Statistical Approaches in the Mechanics of Granular Materials*, editado por S. C. Cowin y M. Satake, Tokio, Japón.

Onoda, G. Y., y E. G. Liniger, 1990, Phys. Rev. Lett. **64**, 2727.

Pak, H. K., E. Van Doorn, y R. P. Behringer, 1995, Phys. Rev. Lett. **74**, 4643.

Pak, H. K., y R. P. Behringer, 1993, Phys. Rev. Lett. **71**, 1832.

Pak, H. K., y R. P. Behringer, 1994, Nature **371**, 231.

Reynolds, O., 1885, Philos. Mag. **20**, 469.

Savage, S. B., y D. J. Jeffrey, 1981, J. Fluid Mech. **110**, 255.

Timoshenko, S. P., y J. N. Goodier, 1970, *Theory of Elasticity*, 3ª ed., McGraw–Hill, pp. 409–422, § 140–142.

Umbanhowar, P. B., F. Melo, y H. L. Swinney, 1996, Nature **382**, 793.

Walton, O. R., y R. L. Braun, 1986, J. Rheol. **30(5)**, 949.

Walton, O. R., y R. L. Braun, 1993, *Joint DOE/NSF Workshop On FLOW OF PARTICULATES AND FLUIDS*, Ithaca, Nueva York.

Warr, S., J. M. Huntley, y G. T. H. Jacques, 1995, Phys. Rev. E **52**, 5583.

Warr, S., y J. M. Huntley, 1995, Phys. Rev. E **52**, 5596.