Simulación de grandes multitudes con dinámica de grupos

Oriam De Gyves[1], Leonel Toledo[1], Iván Rivalcoba[1], Isaac Rudomín[2]

[1] Tecnológico de Monterrey, Campus Estado de México,
Estado de México, México

[2] Barcelona Supercomputing Center,
Barcelona, España

Resumen. La simulación de multitudes ha sido utilizada para crear ambientes urbanos vibrantes e inmersivos, en donde miles agentes virtuales se comportan como peatones reales. Muchos investigadores se han enfocado en simulaciones constituidas únicamente de agentes individuales, sin embargo, en la realidad las multitudes están conformadas no solo de individuos, sino también de pequeños grupos de peatones que viajan juntos. En este trabajo presentamos un sistema que permite simular grandes multitudes, en el hardware gráfico, que además integra la noción de formación de pequeños grupos de agentes. Utilizamos una modificación de la técnica de fuerzas sociales para que los agentes naveguen en el ambiente virtual. Identificamos en ésta un error de formulación que ocasiona que los agentes se atoren bajo circunstancias específicas y proponemos una solución. Los resultados experimentales muestran un cambio en la dinámica de la multitud, y el sistema completo es capaz de simular grandes multitudes de manera interactiva.

Palabras clave: Simulación, grandes multitudes, dinámica de grupos.

1. Introducción

Planeación urbana, educación, seguridad y entretenimiento, entre otras, son algunas de las aplicaciones de la simulación de multitudes hoy en día [3]. Aplicaciones de este tipo pueden utilizarse en lugares públicos para detectar el comportamiento anormal de la población, como en un partido de fútbol o en el sistema subterráneo de una ciudad. Por otro lado, la industria del cine las ha utilizado para crear escenas de grandes batallas o invasiones. Este trabajo se centra en aplicaciones que requieren simulaciones en tiempo real, como el primer ejemplo, por lo que es importante la optimización de los algoritmos utilizados.

Esta área ha sido estudiada por diversos investigadores en la actualidad, sin embargo, es sólo recientemente que se comenzaron a estudiar los grupos que se forman dentro de una multitud. Las simulaciones microscópicas son aquellas que se enfocan en estudiar el comportamiento de los individuos dentro de una multitud. Tradicionalmente, este tipo de simulaciones ha considerado únicamente a los peatones de manera aislada, es decir, que no tienen ningún tipo de relación con algún otro peatón de la multitud. En la realidad, una multitud

está compuesta en su mayoría por pequeños grupos de peatones con algún tipo de relación, como familiares o amigos. Estos pequeños grupos mantienen ciertas formaciones que facilitan la interacción social, lo que cambia la dinámica de una multitud.

Para poder manejar grandes cantidades de agentes, la simulación de los agentes es realizada en el procesador gráfico. Para que el GPU (*Graphics Processing Unit*, unidad de procesamiento gráfico en inglés) pueda trabajar de una manera eficiente, es importante reducir la cantidad de información que se comunica entre éste y el CPU (*Central Processing Unit*, unidad de procesamiento central en inglés), ya que esa comunicación sería por cada cuadro de la simulación. Es por esta razón que utilizamos un método para realizar las búsquedas de proximidad en el GPU, además de los comportamientos, logrando así que toda la simulación se realice en paralelo sin necesidad de compartir información con el CPU por cada cuadro.

La contribución principal de este artículo es el presentar un sistema multiagente capaz de simular grandes multitudes a través de algoritmos altamente paralelos que usan de manera eficiente el procesador gráfico. Se hace uso de técnicas que permiten la simulación de multitudes con la formación de pequeños grupos de agentes virtuales relacionados. Los resultados experimentales indican que la dinámica de la multitud se ve afectada al incluir estas características. La inclusión de estas características podría significar una representación más acertada de la realidad, ayudando a otras áreas a crear aplicaciones más realistas.

El resto del artículo se divide de la siguiente manera. Se presenta una breve discusión sobre el trabajo relacionado en la Sección 2. En la Sección 3 se presenta una técnica diseñada para mejorar el desempeño de las búsquedas de proximidad en tiempo real, lo que permite simulaciones de grandes multitudes. La Sección 4 presenta nuestra formulación para incluir grupos de agentes virtuales en las simulaciones, así como una modificación a la formulación un algoritmo conocido para simulaciones microscópicas. Los experimentos que realizamos para probar nuestro sistema y los resultados son presentados en la Sección 5. Finalmente, se presentan las conclusiones, limitantes y el trabajo futuro en la Sección 6.

2. Trabajo relacionado

La simulación de multitudes es un área que ha sido estudiada de manera extensiva, tanto de manera macroscópica como microscópica. Este trabajo se enfoca principalmente en simulaciones microscópicas, ya que éstas se encargan del estudio de los comportamientos de los agentes, por lo que son más apegadas a la realidad. Esta Sección presenta una breve discusión el trabajo relacionado y del estado del arte.

Craig Reynolds [14] presentó uno de los primeros trabajos de investigación enfocado en comportamientos de agentes virtuales autónomos, llamados *boids*. Este trabajo se encargaba de la navegación de los agentes en base a reglas simples, llamadas *steering behaviors*, y Reynolds las presenta como bloques de

construcción para crear comportamientos complejos. El mismo autor extiende su investigación con comportamientos más complejos [15].

Helbing et al. [10] presenta otro modelo importante para la navegación de los peatones. Este modelo está basado en fuerzas, conocidas como *Fuerzas Sociales*, las cuales hacen referencia a las intenciones de un agente para realizar una acción. De acuerdo con los autores, este modelo es adecuado para peatones en situaciones normales o conocidas. Sin embargo, los mismos autores extienden su investigación para crear un modelo que también es adecuado para situaciones de estrés, al añadir un parámetro de nerviosismo que hace que el comportamiento de los peatones sea más errático y exhiba características de manada [9]. Moussaid et al. [13] extiende el concepto de Fuerzas Sociales para incluir comportamientos y formaciones de grupo entre los agentes. Los autores hacen un estudio sobre el tamaño común de los grupos en multitudes de diferente densidad y cómo éstos afectan la dinámica de la multitud. Millán et al. [12] presenta otra técnica basada en fuerzas, sin embargo, los autores codifican las fuerzas en texturas para poder implementarlo de manera eficiente en el GPU.

De igual manera, existen métodos de navegación geométricos, como los *Velocity Obstacles* introducidos por Fiorini et al. [5]. En este método, los agentes calculan el conjunto de velocidades que los llevaran a una colisión con un obstáculo; para moverse en rutas sin colisiones, los agentes eligen velocidades fuera de este conjunto. Este concepto es aumentado en *Reciprocal Velocity Obstacles* (RVO) por Van den Berg et al. [2] al dividir el esfuerzo de la navegación entre los agentes involucrados. El mismo grupo de investigación optimizó RVO a problemas de programación lineal de baja dimensión, lo que les permitió simular miles de agentes en tiempo real [1].

En todas las técnicas antes mencionadas, existen interacciones entre los peatones, independientemente de si pertenecen o no a un grupo dentro de la multitud. Existen estructuras jerárquicas, como los *octrees*, que subdividen el espacio en regiones que contienen agentes [7]. Una estructura jerárquica muy usada por los investigadores [2,6] es conocida como kd-trees. Recientemente, Dos Santos et al. [17] ha investigado la implementación de estas estructuras en el GPU. Hoff et al. [11] presenta una técnica de detección de colisiones geométrica, rasterizando imágenes y tratando pixeles sobre-escritos como colisiones. De Gyves et al. [4] presenta una técnica basada en diagramas de Voronoi truncado para realizar búsquedas de proximidad de manera eficiente en el GPU. Esta técnica es capaz de superar en desempeño a otras similares, resultando en simulaciones con miles de agentes en tiempo real, utilizando hardware de nivel consumidor.

3. Búsquedas de proximidad

Para poder simular de manera eficiente grandes multitudes, no basta con que la simulación de los comportamientos sea realizada en el GPU, aun cuando se realice en paralelo. Es importante no compartir grandes cantidades de información entre el CPU y el GPU, ya que esto puede afectar de manera negativa el desempeño de una aplicación. En simulación de multitudes, se trabaja

típicamente con miles de agentes virtuales, sin embargo, compartir información entre GPU y CPU en cada cuadro de la simulación para esta cantidad de agentes resulta prohibitiva. Debido a esto, no solo realizamos la simulación de los comportamientos en el GPU, también las búsquedas de proximidad. De esta manera se minimiza la comunicación entre el hardware gráfico y la unidad de procesamiento, resultando en un aumento en el desempeño.

En esta fase del sistema, utilizamos la técnica de diagramas de Voronoi truncado de De Gyves et al. [4]. Esta técnica demostró ser eficiente para el cálculo de búsquedas de proximidad en el GPU, aun cuando este sea considerado como *entry-level* (nivel básico). Dado un conjunto S de sitios en el espacio, un diagrama de Voronoi VD(S) es la descomposición de $\mathbb{R}^d$ en $|S|$ celdas de Voronoi, en donde:

$$V(s,S) = \{p \in \mathbb{R}^d \mid d(p,s) \leq d(p,s') \forall s' \in S\} \tag{1}$$

Por otro lado, un diagrama de Voronoi truncado tVD(S) es la descomposición de $\mathbb{R}^2$ en $|S|$ celdas de Voronoi truncado, en donde:

$$tV(s,S) = \{p \in \mathbb{R}^2 \mid d(p,s) \leq d(p,s') \ \wedge \ d(p,s) \leq r \forall s' \in S\} \tag{2}$$

La Ecuación 1 describe una celda de Voronoi similar a la de la Ecuación 2, con la diferencia de que las celdas en la segunda ecuación están acotadas por un radio r (llamado radio de celda) y no por celdas adyacentes. De esta manera, si la separación entre celdas es mayor a r, el diagrama de Voronoi truncado está compuesto de $|S|$ círculos. Por otro lado, si la separación entre todas las celdas es menor a r, no existe diferencia entre los diagramas.

Al usar estos diagramas, cada una de las celdas representa un agente en la simulación; el diagrama completo representa un mapa de ambiente que cada agente puede usar para obtener, por medio de una búsqueda local, a sus vecinos más cercanos. Se utilizan este tipo de diagramas, en lugar de dibujado tradicional, para que no existan conflictos con el orden de dibujado de las celdas, como se observa en la Figura 1. De igual manera, se garantiza que en caso de que la distancia entre 2 agentes sea menor a r, el área entre estos será distribuida de manera uniforme entre ambos, lo cual es útil para el proceso de muestreo del diagrama.

Para detectar a los vecinos más cercanos usando el mapa de ambiente, cada agente realiza un muestreo en la dirección de su movimiento. Cada agente a realiza un proceso conocido como *Ray Marching* en el diagrama. Comenzando en la posición de a, cada agente lanza n rayos en la dirección de su movimiento, distribuidos de manera uniforme en el área de visión del agente. Cada rayo realiza un muestreo en el mapa de ambiente a ciertos intervalos predeterminados (típicamente menor a r), como se muestra en la Figura 2; en cada muestreo se detecta si en ese punto hay algún otro agente para construir la lista de vecinos.

De Gyves et al. [4] hace un estudio sobre diferentes técnicas para calcular el diagrama. En este trabajo presentamos los algoritmos para generar el diagrama de Voronoi truncado utilizando el procesador gráfico. Como pre-procesamiento se prepara una textura con un gradiente radial. Esta textura es enviada al GPU

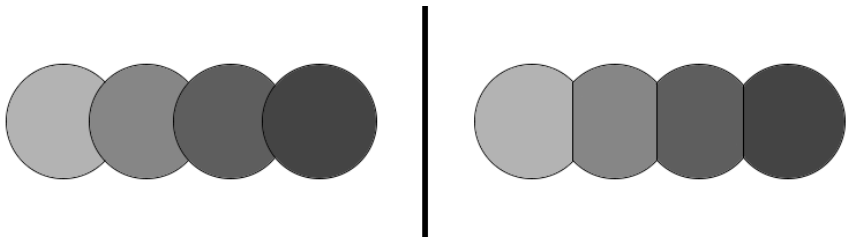

Fig. 1. Cuatro círculos representando a cuatro agentes, dibujados de izquierda a derecha. La imagen de la derecha utiliza diagramas de Voronoi truncado para dibujar las celdas.

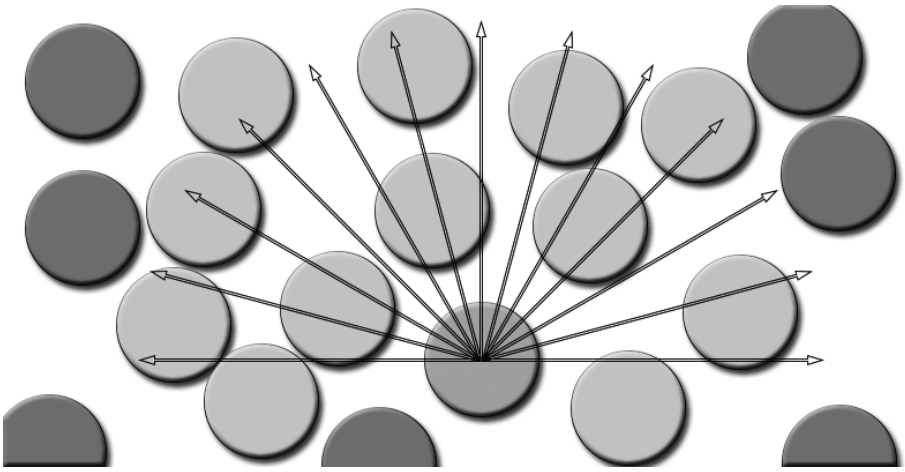

Fig. 2. Muestreo del mapa de ambiente lanzando rayos desde la posición de cada agente en la dirección de su movimiento

y rasterizada en cada celda de Voronoi truncado. La construcción de la textura es realizada en CPU y el Algoritmo 1 presenta los pasos para generarla. Una vez generada la textura, ésta se envía al GPU para poder dibujar el diagrama de Voronoi truncado.

Una vez que la textura con gradiente radial haya sido enviada al GPU, los valores del gradiente se usan como una representación de profundidad del fragmento a rasterizar. De esta manera se utiliza el *Z-buffer* (técnica implementada en todas las tarjetas gráficas actuales para prevenir *Z-fighting*) como un método eficiente para descartar los fragmentos de celdas superpuestas. La implementación de esta técnica es completamente realizada en el GPU, utilizando OpenGL y *GLSL*. El Algoritmo 2 muestra como dibujar cada celda de Voronoi truncado, utilizando una función de prueba de profundidad *GL_LESS*.

Algoritmo 1 Pre-procesamiento. Creación de una textura con un gradiente radial que es utilizada por cada celda de Voronoi truncado.

Requiere: d hace referencia al diámetro de la textura con el gradiente radial.

```
1: función CrearTextura(d)                    ▷ Creación de una textura de diámetro d
2:     tex[d] ← [0, 0, ..., 0]
3:     r ← d/2
4:     de i ← 0 a d realiza
5:         de j ← 0 a d realiza
6:             v ← i − R
7:             h ← j − R
8:             dst ← sqrt(v^2 + h^2)
9:             si dst > r entonces
10:                dst ← r
11:            tex[i × d + j] ← 1 − dst/r
12:    return tex
13: fin función
```

4. Dinámica de grupos

Recientemente han surgido investigaciones enfocadas en incluir grupos de peatones en simulaciones de multitudes. Tal es el caso de Moussaid et al. [13], quien presenta uno de los primeros trabajos que incluye una formulación para crear grupos de peatones dentro de una multitud. Los autores realizan además un estudio sobre las formaciones de los peatones, dependiendo de la densidad de peatones en el área. De acuerdo con sus resultados, los peatones normalmente viajan en formaciones tipo-V, lo que facilita la comunicación e interacción social entre los miembros del grupo.

Algoritmo 2 Dibujado del diagrama de Voronoi truncado utilizando GLSL. Este procedimiento se realiza por cada fragmento a rasterizar de cada celda de Voronoi truncado.

Requiere: tex hace referencia a una textura con gradiente radial. x y y hacen referencia a las coordenadas del fragmento que se va a rasterizar (x, y). c representa el color a dibujar en el fragmento.

```
1: procedimiento DibujarFragmento(tex, c, x, y)   ▷ Ejecutado una vez por cada fragmento.
2:     depth ← 1 − fetch(tex, x, y)               ▷ fetch lee el valor de tex en (x, y)
3:     si depth < f_d entonces   ▷ f_d es el valor de profundidad actual del fragmento
4:         fc = c                                 ▷ f_c es el color actual del fragmento
5:         f_d = depth
6: fin procedimiento
```

Realizamos tomas aéreas de estudiantes caminando en el pasillo de una universidad de la Ciudad de México, como se puede observar en la Figura 3. Los videos fueron grabados en 6 momentos distintos durante un día normal de clases en la universidad. En este estudio observamos a más de 150 personas caminando en varias direcciones, de las cuales el 57 % pertenecía a un pequeño grupo. De acuerdo con Moussaid, hasta el 70 % de los peatones que ellos observaron caminaban en grupos. En ambos casos, la proporción de peatones que viajan en grupos es mayor a la de peatones que viajan solos. De igual manera, en nuestro estudio podemos observar que los grupos de dos a cuatro miembros son los más comunes, mientras que grupos de mayor tamaño se vuelven raros. En nuestro estudio, solo pudimos observar un grupo de más de 4 personas.

Fig. 3. Cinco imágenes de peatones caminando por un pasillo. Como se puede observar, la mayoría de los peatones no camina solo, sino en compañía de otros peatones.

Para que nuestros agentes naveguen por un ambiente virtual, realizamos una implementación en paralelo del trabajo de Moussaid et al. [13]. Como en el caso de las búsquedas de proximidad, los comportamientos de los agentes son implementados en el GPU, utilizando OpenGL y GLSL. Sin embargo, en el caso de los comportamientos, fueron implementados utilizando *Compute Shaders*. Este tipo de shaders permite el uso de cómputo de propósito general utilizando

unidades de procesamiento gráfico (GPGPU o *General Purpose Computations on Graphics Processing Units*), además de permitir el uso de elementos gráficos (la textura con el mapa de ambiente). Rivalcoba et al. [16] ha utilizado una técnica similar para acoplar una multitud virtual con peatones reales capturados en video.

Utilizamos cuatro texturas con los datos requeridos para la simulación: posición, velocidad, grupo y metas. Cada elemento de la texturas (*texel*) i contiene los datos del agente a_i. Para actualizar los valores de las texturas de posición y metas, utilizamos las Ecuaciones 3, 4 y 5.

$$\boldsymbol{f}_i = \boldsymbol{f}_i^d + \boldsymbol{f}_i^n + \boldsymbol{f}_i^o + \boldsymbol{f}_i^g \tag{3}$$

En donde $\boldsymbol{f}_i$ representa la aceleración del agente a_i en el cuadro actual. Esta fuerza es modificada por las motivaciones internas del agente ($\boldsymbol{f}_i^d$), sus vecinos más cercanos ($\boldsymbol{f}_i^n$), los obstáculos más cercanos ($\boldsymbol{f_i}^o$) y la dinámica de grupo ($\boldsymbol{f}_i^g$).

$$\boldsymbol{v}_{inew} = \boldsymbol{v}_i + \boldsymbol{f}_i \cdot \Delta t \tag{4}$$

La nueva velocidad es representada por $\boldsymbol{v}_{inew}$, y se obtiene al sumar la aceleración ($\boldsymbol{f}_i$) con la velocidad anterior ($\boldsymbol{v}_i$).

$$\boldsymbol{p}_{inew} = \boldsymbol{p}_i + \boldsymbol{v}_{inew} \cdot \Delta t + \frac{\boldsymbol{f}_i \cdot \Delta t^2}{2} \tag{5}$$

Finalmente, la posición del agente ($\boldsymbol{p}_{inew}$) se actualiza tomando en cuenta su aceleración y velocidad. En las Ecuaciones 4 y 5, Δt representa el tiempo transcurrido entre dos cuadros de la simulación consecutivos, y se utiliza normalmente en aplicaciones gráficas para obtener velocidades de movimientos independientes del *frame-rate*. Por otro lado, la dinámica de grupo puede ser expresada de la siguiente manera:

$$\boldsymbol{f}_i^g = \boldsymbol{f}^v + \boldsymbol{f}^a + \boldsymbol{f}^r \tag{6}$$

En donde $\boldsymbol{f}^v$ representa la visibilidad del agente. Normalmente se prefiere que los otros miembros del grupo permanezcan dentro del rango de visión del agente. $\boldsymbol{f}^a$ representa la atracción hacia otros miembros de su grupo, representado por el centro de masa del grupo. Finalmente, la repulsión hacia otros agentes, sean o no de su grupo, es representada por $\boldsymbol{f}^r$.

El trabajo de Moussaid está basado en Fuerzas Sociales de Helbing [8]. En este trabajo se describe una *fuerza de cuerpo* y una *fuerza de deslizamiento*, las cuales contrarrestan la compresión de los cuerpos y generan un movimiento relativo tangencial, respectivamente. Al implementar el método, encontramos que la diferencia en la velocidad tangencial ($\Delta \boldsymbol{v}_{ji}^t = (\boldsymbol{v}_j - \boldsymbol{v}_i) \cdot t_{ij}$) usada en la fuerza de deslizamiento es cero ($\Delta \boldsymbol{v}_{ji}^t = 0$) cuando los agentes se mueven en direcciones opuestas con la misma velocidad. Esto resulta en agentes que no pueden evadirse, dejándolos atorados en una línea recta, como puede observarse en la Figura 4.

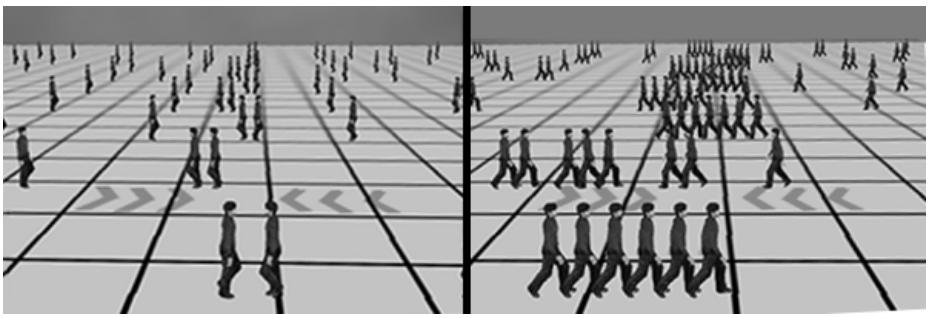

Fig. 4. Los agentes se quedan atorados en una línea recta utilizando la formulación original de Fuerzas Sociales

$$\Delta \boldsymbol{v}_{ji} = \upsilon \tag{7}$$

En la Ecuación 7, υ representa una constante vectorial que satisface $\|\upsilon\| \neq 0$ y es usada en caso de que $\Delta \boldsymbol{v}_{ji} = \boldsymbol{v}_j - \boldsymbol{v}_i \leq \epsilon$. La magnitud de υ es proporcional a la fuerza de deslizamiento, es decir, a mayor fuerza de deslizamiento los agentes se evaden más rápidamente. La prueba anterior fue repetida utilizando esta corrección en la formulación original de la diferencia de velocidad tangencial, con buenos resultados, como se observa en la Figura 5.

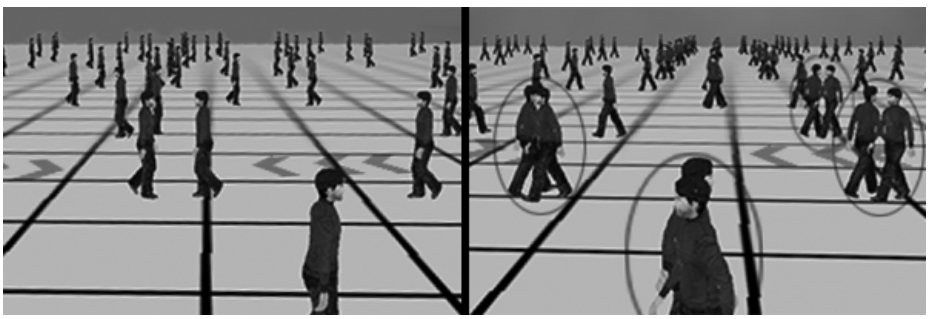

Fig. 5. Con los cambios en la formulación en la diferencia de velocidad tangencial, los agentes pueden evadirse aún bajo las mismas circunstancias que los atoran en una línea recta usando la formulación original de Fuerzas Sociales

5. Experimentos y resultados

Los escenarios que se describen a continuación, así como las técnicas presentadas anteriormente, fueron implementados usando C++ para el código de CPU y OpenGL/GLSL para la simulación y visualización de la multitud en el GPU. Las características de hardware son las siguientes: computadora personal x64 con un procesador Intel i7-2630QM de cuatro núcleos a 2.00 GHz y 8GB de memoria. La tarjeta de video es una nVidia GT540M, la cual es de la familia Fermi y soporta OpenGL 4.3. Esta tarjeta de video es considerada de nivel básico para los estándares actuales.

Realizamos pruebas de evasión de colisiones para verificar que nuestra formulación conservara una correcta navegación de los agentes por el entorno virtual. Esta prueba es conocida como Circle-N (Figura 6), en donde N agentes son colocados en la circunferencia de un círculo y su meta es llegar a la posición diametralmente opuesta. A pesar de que esta prueba no refleja cómo se mueven los peatones en la realidad, es usualmente empleada para probar la navegación de los agentes en una multitud. En esta prueba no se incluye navegación global por la misma razón.

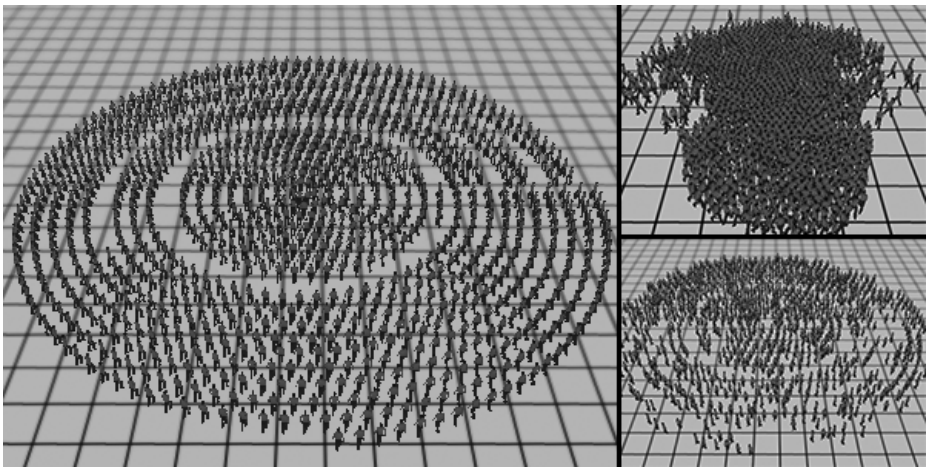

Fig. 6. Prueba de evasión de colisiones. Los agentes inician en la circunferencia de círculos concéntricos y su meta se encuentra en la posición diametralmente opuesta.

El sistema debe ser capaz de simular grandes multitudes (varios miles de agentes) en tiempo real. En la Figura 7 se muestran dos pruebas con hasta 16 mil agentes, simuladas a 60 cuadros por segundo. En estas pruebas se puede observar que la velocidad promedio a la que viajan los agentes, en los escenarios con grupos, disminuye más de 10 % en comparación con los escenarios sin esta característica (de 1.21 m/s a 1.085 m/s). Debido a esta disminución en la velo-

cidad promedio de la población, los agentes tardan más tiempo en llegar a sus metas, ya sea que viajen en grupo o no. Esto refleja el cambio en la dinámica de la multitud cuando se agregan este tipo de propiedades.

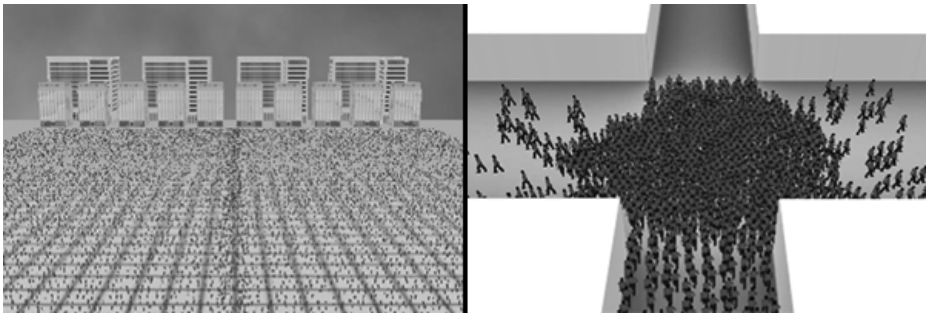

Fig. 7. Pruebas con hasta 16,384 agentes virtuales. Simulaciones en tiempo real (16.6 ms. por cuadro).

La Figura 8 muestra los grupos que se forman en nuestras simulaciones, en donde se han resaltado algunos de los grupos. Con un 4 % de probabilidad, los grupos pueden estar formados por 4 miembros, 12 % por 3 miembros, 40 % por 2 miembros y el resto de los agentes viaja de manera individual. De esta manera, en promedio las simulaciones tienen un mayor número de agentes que viajan en grupo sin que todas las simulaciones tengan las mismas características. Realizamos simulaciones desde 1,024 hasta 16,384 agentes virtuales en tiempo real.

6. Conclusiones, limitantes y trabajo futuro

Las multitudes no son constituidas únicamente de individuos viajando solos; la mayor parte de una multitud se conforma de pequeños grupos de peatones relacionados. Realizamos un estudio para confirmar que la proporción de pequeños grupos normalmente es mayor a los individuos que viajan solos, obteniendo los resultados esperados. Hemos presentado un sistema capaz de simular grandes multitudes que también considera pequeños grupos de agentes relacionados. Este sistema es paralelo e implementado en el hardware gráfico utilizando técnicas de rasterizado y de cómputo general en GPUs. De esta manera se han conseguido simulaciones con miles de agentes en tiempo real.

El sistema de búsquedas de proximidad depende de una textura de ambiente, el diagrama de Voronoi truncado. Este diagrama representa el mundo virtual por donde se mueven los agentes, sin embargo, mientras más grande sea el mundo, más grande debe ser el tamaño de la textura. Esto significa que existe un impacto negativo en el desempeño del sistema conforme mayor sea el tamaño del

Fig. 8. En las imágenes se muestran algunos de los grupos que se forman en nuestro sistema. Los agentes intentan viajar juntos hasta su meta.

mundo. Como trabajo futuro, investigaremos el uso de técnicas de nivel de detalle (específicamente utilizando *mipmaps*) para hacer independiente el tamaño de la textura del tamaño del mundo virtual.

Por otro lado, el sistema de navegación de los agentes está basado en fuerzas. Este tipo de sistemas sufren de problemas cuando las fuerzas de varios agentes se cancelan unas con otras, ocasionando que los agentes no evadan colisiones adecuadamente. Buscaremos trasladar este sistema a un método geométrico, para evitar que sucedan estas posibles fallas. De igual manera, continuaremos investigando la asignación de roles a los miembros de un grupo, como *mamá, amigo, novio, hermana*. La introducción de estos roles podría cambiar también la dinámica de movimiento de una multitud, y acercaría cada vez más estas simulaciones a la realidad.

Referencias

1. van den Berg, J., Guy, S.J., Lin, M., Manocha, D.: Reciprocal n-body collision avoidance. Robotics Research 70, 3–19 (2011)
2. van den Berg, J., Manocha, D.: Reciprocal Velocity Obstacles for real-time multi-agent navigation. In: 2008 IEEE International Conference on Robotics and Automation. pp. 1928–1935. IEEE (May 2008)
3. De Gyves, O., Toledo, L., Rudomín, I.: Comportamientos en simulación de multitudes : revisión del estado del arte. Research in Computing Science. 62(Special Issue: Avances en Inteligencia Artificial.), 319–334 (2013)
4. De Gyves, O., Toledo, L., Rudomín, I.: Proximity Queries for Crowd Simulation Using Truncated Voronoi Diagrams. In: Proceedings of Motion on Games - MIG '13. pp. 87–92. ACM Press, New York, New York, USA (2013)

5. Fiorini, P., Shiller, Z.: Motion Planning in Dynamic Environments Using Velocity Obstacles. The International Journal of Robotics Research 17(7), 760–772 (Jul 1998)
6. Guy, S.J., Chhugani, J., Curtis, S., Dubey, P., Lin, M., Manocha, D.: PLEdestrians: a least-effort approach to crowd simulation. Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation pp. 119–128 (2010)
7. Hadap, S., Eberle, D., Volino, P., Lin, M.C., Redon, S., Ericson, C.: Collision detection and proximity queries. In: Proceedings of the conference on SIGGRAPH 2004 course notes - GRAPH '04. pp. 15–es. ACM Press, New York, New York, USA (2004)
8. Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. Nature 407(6803), 487–90 (Sep 2000)
9. Helbing, D., Farkas, I.J., Molnár, P., Vicsek, T.: Simulation of Pedestrian Crowds in Normal and Evacuation Situations. Pedestrian and evacuation dynamics 21 (2002)
10. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Physical Review E 51(5), 4282–4286 (May 1995)
11. Hoff, K.E., Zaferakis, A., Lin, M., Manocha, D.: Fast and simple 2D geometric proximity queries using graphics hardware. In: Proceedings of the 2001 symposium on Interactive 3D graphics - SI3D '01. vol. 31, pp. 145–148. ACM Press, New York, New York, USA (Feb 2001)
12. Millan, E., Hernandez, B., Rudomin, I.: Large Crowds of Autonomous Animated Characters Using Fragment Shaders and Level of Detail. In: Wolfgang Engel (ed.) ShaderX5: Advanced Rendering Techniques, chap. Beyond Pix, pp. 501—-510. Charles River Media (2006)
13. Moussaïd, M., Perozo, N., Garnier, S., Helbing, D., Theraulaz, G.: The walking behaviour of pedestrian social groups and its impact on crowd dynamics. PloS one 5(4), e10047 (Jan 2010)
14. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. ACM SIGGRAPH Computer Graphics 21(4), 25–34 (Aug 1987)
15. Reynolds, C.W.: Steering behaviors for autonomous characters. In: 1999 Game Developers Conference. pp. 763–782 (1999)
16. Rivalcoba Rivas, J.I., De Gyves, O., Pelechano, N., Rudomín, I.: Coupling Camera-tracked Humans with a Simulated Virtual Crowd. In: Proceedings of the 9th International Conference on Computer Graphics Theory and Applications. pp. 312–321. SCITEPRESS - Science and and Technology Publications (2014)
17. dos Santos, A.L., Teixeira, J.M.X., de Farias, T.S., Teichrieb, V., Kelner, J.: kD-Tree Traversal Implementations for Ray Tracing on Massive Multiprocessors: A Comparative Study. In: 2009 21st International Symposium on Computer Architecture and High Performance Computing. pp. 41–48. IEEE (Oct 2009)