Simulació de models de físiques en OpenGL i shaders

Kirian Careta Freixa

Resum—En els darrers anys les capacitats gràfiques computacionals han augmentat considerablement, permetent l'ús d'objectes 3D en temps real i, fins i tot, efectes complexes tals com explosions o moviment de líquids. Aquests elements necessiten d'uns entorns gràfics específics per ser implementats i OpenGL ho permet. Amb la realització d'aquest projecte es busca indagar en aquests entorns, es busca aprendre a utilitzar OpenGL per programar elements complexes tals com els sistemes de partícules, recursos de gran potencial i flexibilitat capaços de simular tot tipus de comportaments a utilitzar en tot tipus d'àmbits (videojocs, cinema, simulacions científiques, etc.). En el projecte es desenvoluparan dos sistemes de partícules (una font d'aigua i un castell de focs) utilitzant diferents tipus d'algorismes i diferents tècniques per tal de realitzar anàlisis i extreure'n conclusions. Es buscarà, també, realitzar els desenvolupaments per permetre l'encapsulament i la seva importació en entorns 3D externs. Es tindrà en compte els diferents mètodes de test a utilitzar en el projecte, així com la parametrització dels diferents sistemes per permetre la modificació en els comportaments de forma simple i còmode.

Paraules clau—OpenGL, partícules, shaders, GLSL, Visual Studio, simulació, parametrització, 3D, font, castell de focs, GLEW, GLUT, vèrtex, array de vèrtexs.

Abstract—In recent years computational graphic skills have greatly incleased, allowing the use of 3D objects in real time, even complex effects such as explosions or movement of liquids. These elements need to be implemented in specific graphical environments and OpenGL allows it. With the realization of this project seeks to investigate these environments to learn how to use OpenGL to program complex elements such as particle systems, resources with high potential and flexibility capable of simulating all types of behaviours to be used in all kinds of areas (videogames, cinema, scientific simulations, etc.). In the project will develop two particle systems (a water fountain and a firework) using different algorithms and different techniques to perform analysis and draw conclusions. Will seek also make development to allow encapsulation and import in externs 3D environments. Also will take into account the different test methods used in the project, as well as the parameterization of the different systems to allow modifications in the behaviour so simple and comfortable.

Index Terms—OpenGL, particles, shaders, GLSL, Visual Studio, simulation, parameterization, 3D, fountain, firework, GLEW, GLUT, vertex, vertex array.

1 Introducció

En gràfics per computador cada cop es tendeix més a representar els objectes 3D en temps real, tant de forma estàtica com dinàmica. En la part dinàmica una de les darreres tendències és la representació de fenòmens de partícules (tals com explosions, moviment de líquids, simulació de fogueres, efecte de fum, etc.) que van un pas més enllà: combinar-ho amb altres elements en escenes 3D creant entorns realistes i amb dinamisme.

La complexitat d'aquests fenòmens a l'hora d'implementar-ho en computadors ha vingut complementada pel fet que les plaques gràfiques han adquirit potència de processament i, també, per l'aparició dels shaders, petits codis de programació sobre arquitectures paral·leles que potencien de forma notable els resultats a obtenir i poden afegir grans millores tant estètiques com de rendiment.

Amb aquest projecte es pretén indagar en aquests sistemes utilitzant la llibreria OpenGL amb shaders per desenvolupar i analitzar diferents fenòmens realistes usant varis algorismes de programació. Es busca estudiar la creació i desenvolupament de sistemes de partícules i aprendre a utilitzar aquests elements com a recursos visuals en entorns 3D aplicant diferents algorismes de desenvolupament.

En els següents apartats es comentaran, primerament, l'especificació dels objectius per tractar, posteriorment, les eines a utilitzar i la metodologia a seguir.

Quan algú escolta parlar sobre el concepte de sistemes de partícules dins el món de la informàtica s'imagina tot tipus d'elements visuals, amb tot tipus d'efectes i de comportaments. Certament el concepte és complex i pre-

- *E-mail de contacte: Kirian.Careta@e-campus.uab.cat*
- *Menció realitzada: Computació*
- *Treball tutoritzat per: Enric Martí Gòdia (Ciències de la Computació)*
- *Curs 2013/14*

senta certa dificultat d'especificació respecte la seva realització i desenvolupament, i aquest fet és la principal motivació d'aquest projecte.

1.2 Objectius

Els principals objectius del projecte són els següents:

1. Estudiar i analitzar els algorismes de sistemes de partícules actuals, com són els de simulació de líquids, representació d'explosions, efecte de foguera o comportament de fum.

2. Realitzar comparacions entre algorismes de programació. L'ús d'un o altre algorisme en el desenvolupament del sistema de partícules pot provocar diferents resultats en rendiment i cost.

3. Implementar algorismes de programació parametritzables en OpenGL i GLSL. Mitjançant la parametrització en els algorismes es busca obtenir canvis en els comportaments dels sistemes de forma simple a partir d'una variació en els paràmetres que hi intervinguin.

4. Realitzar mesuraments de costos i esforç d'execució de diferents sistemes a partir de diferents parametritzacions.

5. Implementar els sistemes de partícules de forma encapsulada per permetre la seva importació en escenes 3D externes.

Per dur-ho a terme, però, es necessitarà l'ús d'eines i entorns específics per a tal desenvolupament, que es detallen a continuació.

1.3 Eines i metodologies

Les eines principals i més importants són:

- **OpenGL**: [5] Llibreries gràfiques de gran ús a nivell mundial pel desenvolupament d'aplicacions visuals. Permet la realització multiplataforma de sistemes de partícules.

- **Shaders**: Algorismes de tractament gràfic amb un gran potencial i moltes opcions d'aplicació. Utilitzen el potencial de la GPU de l'equip informàtic al permetre execució en paral·lel.

- **GLSL**: [6] Llenguatge utilitzat en l'ús de shaders per a OpenGL.

- **Visual Studio**: Entorn de programació compatible amb les llibreries OpenGL en Windows. Requereix d'una configuració prèvia del projecte.

- **Llenguatge C++**: Llenguatge de programació a utilitzar junt amb OpenGL. Ofereix una gran flexibilitat i potencial de desenvolupament.

Juntament amb elles es seguirà una metodologia de treball variant del procés en **cascada** per avançar en el projecte. Aquesta consta de les següents fases o etapes:

1. **Iniciació** o etapa de recollida d'informació.

2. **Disseny** previ al desenvolupament.

3. **Implementació** i parametrització.

4. **Testeig** de la feina realitzada.

5. **Anàlisi** dels resultats obtinguts.

6. **Finalització** amb la importació a entorns 3D.

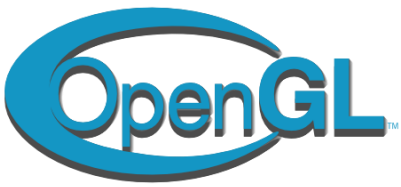


Fig. 1 Logotip de les llibreries OpenGL.

2 DESENVOLUPAMENT

La metodologia de realització del projecte inclou vàries etapes (comentades anteriorment) a desenvolupar per arribar als objectius presentats.

2.1 Recollida d'informació

En la recollida realitzada s'han analitzat dos conceptes principals (junt amb informació d'OpenGL, de tractament de shaders, etc.): tipus d'algorismes de programació de sistemes de partícules, i tipus de comportaments dels mateixos, dels quals s'han extret els següents resultats:

Comportaments dels sistemes:

- Simulació d'una font d'aigua (o líquid).
- Simulació d'una cascada.
- Simulació d'una explosió.
- Simulació de fum.
- Simulació de foc.
- Representació d'energia abstracta.

Tipus d'algorismes:

- Tractament individualitzat de vèrtexs via OpenGL.
- Utilització d'arrays de vèrtexs.

Realment hi ha un gran catàleg d'opcions i combinacions a desenvolupar, i la veritat és que algunes d'altres s'han quedat en el tinter. El que està clar és que aquesta recerca ens produeix una guia a seguir en el treball, ja que defineix i concretitza les opcions disponibles.

2.2 Disseny dels sistemes

Observant les opcions concretades en l'apartat anterior es seleccionaran dos sistemes de partícules a desenvolupar utilitzant els dos tipus d'algorismes de programació i, a més, aplicant shaders per donar efectes de llum i color. Abans, però, cal definir i especificar de què tracta un sistema de partícules. Un sistema de partícules és un conjunt de punts (o partícules) que evolucionen en el temps modificant el seu comportament i el seu estat (posició, velocitat, color, etc.), generant així efectes i simulacions de tot tipus.

Hi ha un conjunt de comportaments en comú en els sistemes de partícules a mode de requisits pel seu correcte desenvolupament:

- Les partícules evolucionen i canvien durant el pas del temps.
- Les partícules disposen d'una posició i d'una velocitat específica en cada instant de temps.
- Les partícules "reneixen" o es reinicien una vegada la seva vida útil ha arribat al final.

Per tant, és evident que en el sistema existirà un "mòdul" de creació de les partícules i un d'evolució de les mateixes, per així poder crear el comportament desitjat i arribar a la simulació buscada.

Així doncs, en l'execució del sistema de partícules, aquestes evolucionen i canvien amb el pas del temps, modificant així posició i velocitat. Aquesta evolució, però, s'acaba quan la vida útil de la partícula s'esgota (per poder donar lloc a un reinici del moviment de les partícules i així crear el comportament de forma contínua). Es pot afirmar, doncs, que les partícules tenen un atribut referent a la seva vida en l'execució amb el que, al acabar-se, la seva posició es reinicia i torna a "néixer" en el punt d'origen comú per repetir l'evolució i el recorregut.

El moviment que realitzen les partícules segueix la següent fórmula matemàtica:

$$Position_{t+1} = Position_t + (Velocity_t * UpdateVel) \qquad (1)$$

El que significa aquesta fórmula (1) és que hi ha diferents elements importants a tenir en compte en les iteracions en que evoluciona el sistema. En cada espai de temps hi ha un valor de posició en l'espai (x,y,z) i un valor de velocitat actual (sx,sy,sz) on, en aquest cas, sobresortirà el valor de velocitat en l'eix Y. El darrer element en la fórmula, *UpdateVel*, és un recurs de parametrització que es comentarà en l'apartat corresponent a la implementació.

El primer sistema de partícules a implementar ha de ser suficientment bàsic com per aprendre amb ell el seu funcionament, conèixer la seva creació i entendre el seu comportament. Però, també, ha de presentar el suficient potencial com per experimentar amb ell i servir de referència. Així doncs, el comportament pel sistema de partícules que millor encaixa en els requisits comentats és el **sistema de la font d'aigua**, ja que permet diverses parametritzacions i variacions molt interessants per a posteriors anàlisis.

Una vegada decidit i seleccionat el tipus de sistema de partícules, tenint en compte que s'utilitzaran els diversos tipus d'algorismes disponibles, cal planificar el funcionament que dóna lloc al seu comportament. En el cas del sistema de la font d'aigua, les partícules realitzen un moviment vertical positiu partint d'un punt d'origen comú per, posteriorment, caure efecte de la gravetat una vegada hagin arribat al punt de màxima altitud.

Encara falta, però, seleccionar el segon sistema de partícules a realitzar. Tenint en compte que la font d'aigua és el sistema de partícules escollit per utilitzar com a referència i aprendre amb ell, el segon sistema ha realitzar ha de presentar un canvi radical en tots els sentits però, a la vegada, comparteixi les bases definides. El sistema escollit, doncs, és una variant del comportament de les explosions: un **castell de focs** (o focs d'artifici). És una simulació molt interessant i visualment espectacular, que servirà per aplicar els conceptes apresos amb la font d'aigua i afegir, per què no, alguna nova tècnica.

Abans, però, s'ha d'implementar el primer sistema de partícules seleccionat i anar pas a pas.

2.3 Implementació

En aquest projecte s'utilitzaran les llibreries GLEW i GLUT d'OpenGL:

- GLEW (OpenGL Extension Wrangler Library) [7] és una extensió d'OpenGL, open-source, que proveeix mecanismes eficients en temps d'execució per determinar les extensions d'OpenGL disponibles en la plataforma que s'està utilitzant i, a més, permet l'ús dels shaders. És compatible amb Windows, Linux, Mac, FreeBDS, Irix i Solaris.
- GLUT (OpenGL Utility Toolkit) [8] és una biblioteca d'utilitats per OpenGL que proveeix funcions d'entrada-sortida amb el sistema operatiu. És, també, multiplataforma, però no open-source.

Una vegada incloses les llibreries en la classe, ja es poden utilitzar les funcionalitats que proveeixen. El següent pas consisteix en definir la funció *main* de la classe principal del projecte, la funció que prepararà la configuració per a una correcte execució. Aquesta preparació la realitza aquest conjunt de funcions de GLUT:

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DEPTH |
GLUT_DOUBLE | GLUT_RGBA);
glutInitWindowPosition(WINDOW_X,
WINDOW_Y);
glutInitWindowSize(SIZE_X, SIZE_Y);
glutCreateWindow("Nom projecte");
```

Què s'està realitzant amb aquestes funcions? Amb *glutInit* s'està iniciant la llibreria GLUT, i amb *glutInitDisplayMode* s'està configurant la visualització en pantalla. La resta de funcions s'utilitzen per generar i configurar una finestra on mostrar el contingut de l'execució. Es defineix la posició en pantalla on es mostrarà la finestra, les seves dimensions i el seu títol.

L'execució, però, no funcionarà només amb aquestes funcions comentades. Falta indicar la funció amb la qual es visualitzarà l'execució i, molt important, indicar el timer d'actualització d'aquesta. Aquestes funcions es poden dur a terme mitjançant:

```
glutDisplayFunc(FuncioVisualització);
glutTimerFunc(4, timerCallBack, 1);
```

El paràmetre *4* indicat en la funció de timer és el valor en milisegons que indica cada quan s'actualitzarà. La resta de paràmetres són necessaris en la funció i no tenen major rellevància. Aquesta funció, però, treballa en conjunt amb *glutPostRedisplay()*, crida per actualitzar la pantalla de visualització.

Hi ha també més funcions interessants que poden ser utilitzades junt amb les dues darreres comentades, com són la funció per redimensionar la finestra o la funció per l'ús del teclat en l'execució, entre d'altres.

Tenint en compte la informació especificada sobre els sistemes de partícules, la millor forma d'assignar a les partícules atributs de posició, velocitats, etc. és definint-ho en un registre (o struct), de manera que resulta còmode i senzill realitzar les evolucions amb el pas del temps a mida que les iteracions avancin.

```
Struct PARTICLE {
      float x, y, z;
      float sx, sy, sz;
      float age;
} particle[MAX_PARTICLES];
```

Els valors (x,y,z) corresponen a la posició de la partícula, mentre que (sx,sy,sz) correspon a la seva velocitat en aquell instant de temps. També és fonamental tenir en compte l'atribut *age*, per poder saber el seu recorregut en el sistema. Un cop creat el struct amb la informació de la partícula, aquest s'ha d'assignar a alguna variable per poder utilitzar-ho. Per a això s'ha creat un array amb nom *particle* que representarà l'estat de totes les partícules del sistema, i aquest array té MAX_PARTICLES posicions, definit com a 50000.

Les funcions principals del sistema són: *create*, *evolve*, i *draw*. El primer a desenvolupar és la funció de visualització, *draw*, per tal de situar el sistema i veure millor quan executar les altres dues funcions. Per dibuixar en pantalla en OpenGL hi ha les funcions *glVertex* que proporcionen molta flexibilitat a l'hora de fer-ho, ja que printen un punt que pot ser configurat en diferents tipus d'elements en pantalla (un punt, un triangle, un polígon, etc.). En aquest cas es poden utilitzar punts, GL_POINTS, per començar a tractar el sistema. Una primera versió de la funció de visualització seria:

```
for (int i=0; i<MAX_PARTICLES; i++){
  glBegin(GL_POINTS);
  glVertex3f(p[i].x, p[i].y, p[i].z);
  glEnd();
}
```

Utilitzant aquest bucle en la funció es dibuixaran en pantalla els diferents punts que componen el sistema de partícules (on *p[i]* equival a *particle[i]*). Però, primer cal crear els punts, i també anar-los evolucionant perquè existeixi un moviment en pantalla, un comportament. Si s'ha decidit que l'atribut de l'edat / vida és que proporciona una referència en aquella partícula, una bona opció és comparar aquest atribut per veure si s'ha esgotat i, de ser així, reiniciar la partícula. I, si la vida no s'ha esgotat, actualitzar la partícula. Per tant, el bucle anterior quedaria de la següent forma:

```
for (int i=0; i<MAX_PARTICLES; i++){
  if (p[i].age < 0) create(i);
  else{
      glBegin(GL_POINTS);
      glVertex3f(p[i].x, p[i].y, p[i].z);
      glEnd();
  }
  evolve(i);
}
```

Encara falta tenir en compte que les partícules poden haver realitzat el recorregut sense esgotar la seva vida útil, on s'hauria de reiniciar d'igual manera (per exemple, es pot utilitzar com a comparador la distància en l'eix Y. Les partícules neixen en el punt d'origen i es pot comparar que quan hagin realitzat el recorregut i, al caure, sobrepassin el punt d'origen en l'eix Y, es reiniciïn).

Amb la funció de visualització implementada i amb els punts de creació i evolució definits ja es pode començar a desenvolupar les respectives funcions: *create* i *evolve*.

- Create:

```
p[i].age = random();
p[i].x = ORIGIN_X;
p[i].y = ORIGIN_Y;
p[i].z = ORIGIN_Z;
p[i].sx = random();
p[i].sy = random();
```

```
p[i].sz = random();
```

- Evolve:

```
p[i].age = p[i].age - AGE_DECREMENT;
p[i].x = p[i].x + (p[i].sx * UPDATE_VEL_X);
p[i].y = p[i].y + (p[i].sy * UPDATE_VEL_Y);
p[i].z = p[i].z + (p[i].sz * UPDATE_VEL_Z);
p[i].sy = p[i].sy – VEL_DECREMENT;
```

En la funció de creació (*create*) de partícules es pot veure com molts dels valors són valors a l'atzar (*random*), però dins d'un rang (per exemple, [0,1]). El rang de *age* serà diferent al de les velocitats, on la velocitat en Y serà superior a X i Z. Pels valors per defecte en la posició es poden utilitzar constants definides.

En la funció d'evolució (*evolve*) hi intervé la fórmula definida anteriorment (1), ja que és la base en l'actualització de les posicions. D'altra banda, la vida útil de la partícula disminueix, igual que passa amb la velocitat vertical en cada iteració temporal.

Amb totes les funcions implementades es pot obtenir un resultat com el que es mostra en la figura 2.

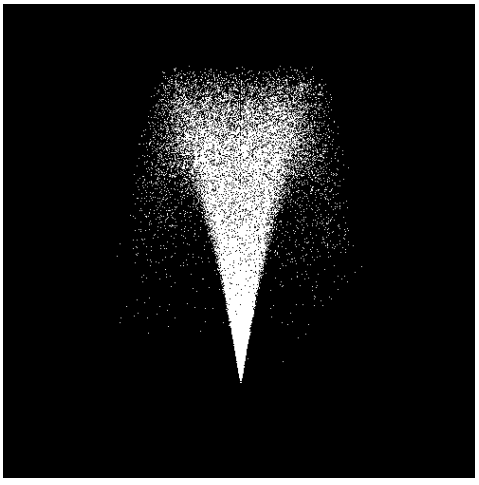


Fig. 2 Sistema de partícules d'una font d'aigua.

Aquest sistema de partícules és una bona base per aplicar-hi optimitzacions i detalls que millorin visualment al sistema. Hi ha dos optimitzacions a realitzar:

- Les partícules actuals no mostren color, es dibuixen blanques, i aplicant color milloraria notablement l'efecte de font. OpenGL proveeix funcions per a tal fi, com és *glColor*, amb la que es poden pintar els *glVertex* que es dibuixin. Un bon color que es pot aplicar en aquest cas és (0.7,0.7,1.0), que equival en nomenclatura RGB a un color blau d'intensitat mitjana. El color variarà en funció del pas del temps a mida que les partícules evolucionin.

- Les partícules representades com a punts poden visualitzar-se sense la força suficient per donar consistència al sistema. Això es pot arreglar representant-les com a GL_QUADS a partir de quatre punts per partícula (amunt-esquerre, amunt-dreta, avall-esquerre, i avall-dreta).

- Es poden afegir detalls en l'escena per representar millor l'efecte 3D i millorar la presència del sistema de partícules. Un exemple d'això és dibuixar una graella a mode de terra, per situar millor en l'escena els elements que hi hagi. Es pot mostrar de color gris, per diferenciar-ho de la resta.

El resultat d'aplicar aquestes optimitzacions es mostra en la figura 3.

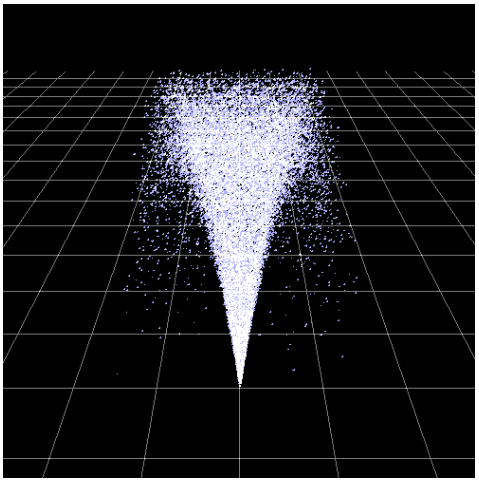


Fig. 3 Sistema de partícules amb les optimitzacions aplicades.

Bé, el primer sistema de partícules està implementat, però s'ha utilitzat un dels dos tipus d'algorismes especificats anteriorment. Falta realitzar la font utilitzant array de vèrtexs, el qual funciona de forma diferent. És necessari que les posicions i el color de les partícules es carreguin via arrays en la funció de mostrar per pantalla específica del tipus d'algorisme d'array de vèrtexs (*glDrawArrays*). Aquests dos arrays (de mida igual al número de partícules a mostrar en pantalla) s'han d'activar en la funció, i desactivar posteriorment del seu ús. Per tant, per utilitzar aquest tipus d'algorisme s'ha de substituir el bucle definit anteriorment per:

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(3,GL_FLOAT,0,ArrayPosicions);
glColorPointer(3,GL_FLOAT, 0,ArrayColors);
glDrawArrays(GL_QUADS,0,MAX_PARTICLES*4);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```

Mitjançant les dues primeres funcions i les dues últimes el que es fa és activar i desactivar l'ús dels arrays de posicions i colors. Amb *glVertexPointer* i *glColorPointer*

indiquem quins són els dos arrays a utilitzar, de quin tipus són, i quants elements contenen (en aquest cas són floats i tenen tres elements per cada partícula ((x,y,z) per la posició, i (r,g,b) pel color). Per últim, en la funció *glDrawArrays* s'especifica el tipus d'elements a visualitzar (GL_QUADS) i el número d'elements a tractar dels arrays utilitzats (com que pels GL_QUADS es necessiten quatre punts per cada partícula la mida és MAX_PARTICLES*4).

S'han utilitzat els dos tipus d'algorismes de programació per desenvolupar la font, però encara falta el recurs dels shaders. Per a què utilitzar-los? Els shaders proporcionen la opció de poder tractar molts aspectes en pantalla (il·luminació, color, moviment, etc.) i, en aquest cas, s'utilitzaran per modificar el color del sistema, optimitzant el resultat aconseguit per apropar-se més al realisme en la visualització.

Hi ha dos tipus de shaders fonamentals en l'aplicació de shaders en el projecte: vèrtex shader i fragment shader. El primer permet tractar directament els diferents vèrtexs en pantalla mentre que, el segon tipus, permet tractar diferents fragments que hi hagin, ja siguin triangles, polígons, etc. Per modificar el color, certament, es poden usar qualsevol dels dos, però en aquest projecte s'utilitzarà el fragment shader, ja que s'utilitzen GL_QUADS en lloc de punts. També, s'utilitzarà un nou atribut en les partícules per tal de donar una major comoditat al shader per assignar color a cadascuna d'elles: el temps que porten actives. Cada partícula realitza un recorregut ascendent, fet el qual resulta interessant per assignar color més vius com més amunt es troba la partícula.

L'ús de shaders no queda centralitzat en els sistemes de partícules, ja que afecten a tota la informació en pantalla i, això, afecta també a la graella que s'utilitza com a terra. Per tant, es transmetrà informació cap als shaders indicant el tipus d'elements a tractar (per la graella s'utilitzarà un tipus 0 i, pel sistema, un tipus 1) de forma que es podran diferenciar i tractar de forma personalitzada. El funcionament i aplicació dels shaders serà comentat en el pròxim apartat.

El resultat que s'obté després del tractament es mostra en la figura 4.

En canvi, el comportament del sistema del castell de focs és completament diferent, ja que aquest es composa de dues fases molt diferenciades:

- Fase d'ascens vertical, on totes les partícules es troben concentrades en un mateix punt.

- Fase d'explosió, on els punts es disparen en totes direccions un cop s'arriba a una alçada determinada.

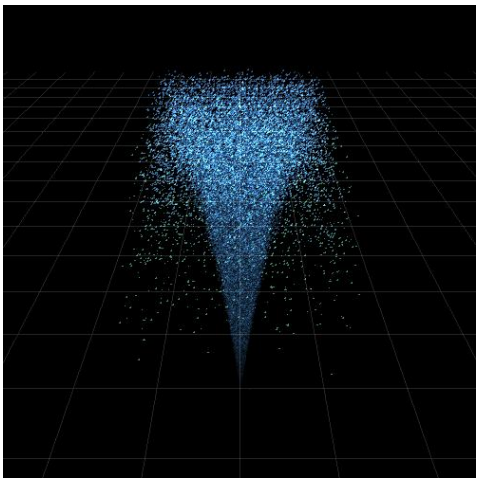


Fig. 4 Sistema de partícules de la font amb shaders.

Per a realitzar aquest comportament es pot utilitzar l'estructura de les partícules definida pel primer sistema implementat, ja que la base és la mateixa. Però, tot hi així, s'han de modificar una mica els atributs i canviar l'enfoc. En el sistema anterior havia un sol sistema, i els atributs anaven en funció d'ell. En canvi, amb aquest sistema de partícules es desenvoluparà un castell de focs complet, amb varis sistemes de partícules dins, i això comporta que els valors de posició i velocitat siguin arrays de la mida del número de partícules a mostrar. L'estructura de les partícules seria, doncs, així:

```
Struct FIREWORK {
     float x, y, z[FIREWORK_PARTICLES];
     float sx, sy, sz[FIREWORK_PARTICLES];
     float particleSize;
     bool hasExplode;
} particle[FIREWORKS];
```

Els atributs *particleSize* i *hasExplode* són per saber si la mida de les partícules del sistema en tractament (ja que hi haurà castells de focs amb partícules més grans que d'altres, per donar varietat i realisme) i per saber si ha arribar al punt d'explotar.

La resta funciona d'igual manera, amb funcions de creació i evolució (junt amb una de nova d'explosió, que canviar el comportament de les partícules afegint velocitats altes en cada eix).

El resultat que s'obté fins ara es mostra en la figura 5.

Hi ha el problema, com es pot veure, que és difícil distingir el que està passant, ja que un detall característic dels castells de focs és l'estela que deixen i, en aquest sistema no n'hi ha. N'hi pot haver? En OpenGL existeix una funció anomenada *glAccum* que el que fa és precisament aquest efecte, d'estela, actuant amb un buffer acumulatiu. Utilitzant aquest recurs d'OpenGL s'obté un resultat més optimitzat, molt més visual, tal i com es mostra en la figura 6.

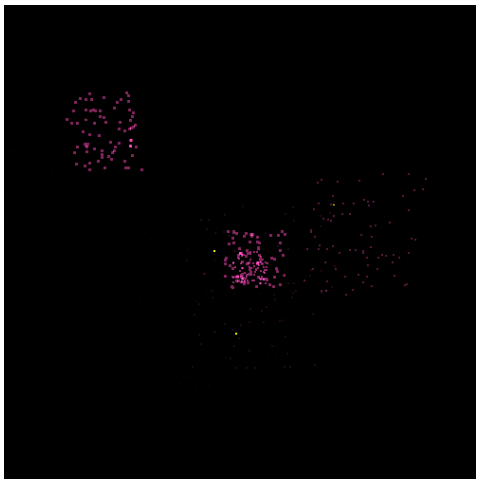


Fig. 5 Sistema de partícules del castell de focs.

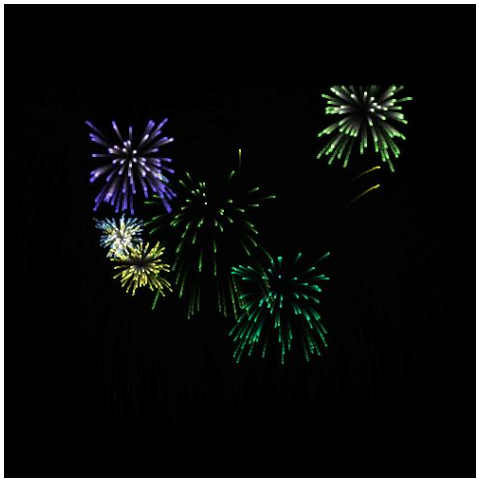


Fig. 6 Sistema de partícules del castell de focs optimitzat.

Arribats a aquest punt ja estan els dos sistemes de partícules implementats, i s'han utilitzat els dos tipus d'algorismes especificats inicialment. A més a més s'han aplicat shaders sobre la font per aconseguir una millora notable del resultat final. Així doncs, ja es pot passar al següent punt a tractar en el projecte: el testeig de la feina realitzada.

2.4 Shaders

Durant la implementació dels sistemes de partícules s'han utilitzat shaders per aplicar color i donar un resultat més visual. Els shaders són un recurs complex sobre el que hi ha molt a dir, però hi ha una sèrie de conceptes d'importància en el seu funcionament:

- Hi ha dos tipus principals de shaders, vèrtex shader i fragment shader, que serveixen per tractar els diferents vèrtex que hi ha en pantalla i els diferents fragments, respectivament. Quan es parla de fragments es refereix a conjunt de vèrtexs, com pot ser un triangle, un polígon, etc.

- Darrere de l'ús de shaders hi ha un procés de càrrega dels mateixos, pel qual s'usen els objectes *Program*. Un programa pot carregar shaders i aplicar-los en l'execució.

- Els shaders treballen en nivell de GPU, permetent la paral·lelització del que estiguin executant, però pot necessitar que se li indiquin certs atributs per funcionar correctament (dels quals n'hi ha de varis tipus). En el cas pràctic d'aquest projecte, se li ha hagut d'indicar el tipus d'element que s'estava mostrant en pantalla (sistema de partícules o graella del terra) per poder-los diferenciar.

En el projecte s'utilitza principalment el vèrtex shader per aplicar els canvis, ja que al mostrar-se punts en el sistema és més pràctic tractar-ho com a vèrtexs.

2.5 Test

En el testeig s'han realitzat les següents comprovacions:

- Test de caixa blanca en les funcions per assegurar que realitzen el que s'espera d'elles i treballen amb les dades de forma correcte. Per exemple, en la funció d'evolució les partícules han de créixer correctament o, en la de creació, s'han d'inicialitzar tots els valors segons s'espera.

- Test de caixa negra amb les funcions, per verificar que retornen el que s'espera d'elles.

- Proves unitàries en els diferents elements en el projecte (*main*, diferents sistemes, etc.) per verificar que el seu comportament és el correcte.

- Proves d'integració per verificar que el conjunt funciona correctament.

- *Test driven development* a partir del que se sabia que havien de realitzar algunes funcions. És important conèixer prèviament el resultat a obtenir amb una funció per poder enfocar correctament el desenvolupament de la mateixa.

Amb aquestes proves s'ha validat que els funcionaments de les funcions es comporten correctament i el sistema té el funcionament esperat.

El test, realment, s'ha anat realitzant de forma paral·lela a la implementació, ja que és important verificar el funcionament d'una part per poder desenvolupar amb tranquil·litat la següent.

Una vegada ha estat el sistema implementat s'ha realitzat un test final de verificació del comportament, el qual consisteix en executar el sistema durant un període de temps significatiu (30 segons – 1 minut) per comprovar que funciona correctament.

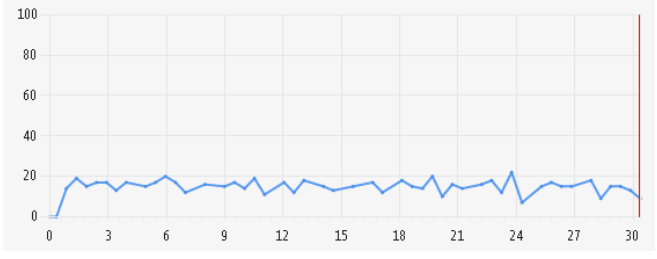
2.6 Anàlisi

En l'anàlisi dels sistemes desenvolupats es duran a terme mesuraments sobre el rendiment de CPU en l'execució, sobre l'ús que se li dóna, pretenent observar el rendiment de cada algorisme i la càrrega que genera sobre la CPU. Per a realitzar-ho s'utilitzaran execucions de 30 segons a partir dels següents recursos:

- Sistemes de partícules font i castell de focs.
- Sistemes de partícules font i castell de focs amb array de vèrtexs.
- Processador Intel Core i7-3610QM 2.3GHz.
- Targeta gràfica Intel HD Graphics 4000.
- Targeta gràfica NVIDIA GeForce GT 640.

El primer anàlisi realitzat serà a partir del sistemes de partícules de la font (amb array de vèrtexs i sense) utilitzant les dues targetes gràfiques. Es mesurarà el percentatge d'ús de CPU en execució i s'especificarà el principal recurs utilitzat en ella (ja sigui funció de l'algorisme, drivers gràfics, etc.), així com el percentatge d'ús d'aquest recurs. El número de partícules a mostrar és el per defecte en els sistemes (50000 per la font i 75 pel castell de focs).

(a)



(b)

(c)

(d)

Fig. 7 Rendiment de CPU amb l'execució del sistema de partícules de la font per les diferents targetes gràfiques. Font amb Intel (a), font amb NVIDIA (b), font amb array i Intel (c), font amb array i NVIDIA (d).

TAULA 1
ANÀLISI DE LA FONT

	Font Intel	Font (array) Intel	Font NV	Font (array) NV
Mitjana ús CPU	~18%	~16.5%	~15.5%	~13%
Principal recurs usat	Drivers gràfics	Funció d'actualització	Drivers gràfics	Funció d'actualització
Percentatge	63.9%	57.8%	62.3%	58.9%

Els resultats que s'obtenen dels mesuraments són interessants i, dintre de tot, esperats. Quan s'utilitza la targeta gràfica NVIDIA, més potent que la Intel, el percentatge d'ús de CPU disminueix, fet que significa que hi ha menys càrrega en l'execució i, per tant, es realitza de forma més lleugera, més ràpida. També es pot apreciar un gran canvi al usar array de vèrtexs o no. Utilitzar array de vèrtexs s'optimitza considerablement la funció de visualització, i el principal recurs utilitzat és el d'actualització de les partícules. En canvi, al no utilitzar array de vèrtexs la funció de visualització produeix una gran càrrega i pren importància en l'execució al convertir-se en el recurs més usat (drivers gràfics).

L'anàlisi de la figura 8 és sobre el segon sistema de partícules, el castell de focs:

(a)

(b)

(c)

(d)

Fig. 8 Rendiment de CPU amb l'execució del sistema de partícules del castell de focs per les diferents targetes gràfiques. Focs amb Intel (a), focs amb NVIDIA (b), focs amb array i Intel (c), focs amb array i NVIDIA (d).

Es pot veure, novament, que els gràfics en els mesuraments són molt semblants entre els quatre anàlisis realitzats, de manera que el rendiment no varia de forma molt contundent. Però, al mesurar el sistema del castell de focs (el qual utilitza la tècnica del *glAccum*), s'obtenen uns resultats interessants a diferència dels comentats anteriorment amb la font, els quals es mostren en la taula 2.

TAULA 2
ANÀLISI DEL CASTELL DE FOCS

	Focs Intel	Focs (array) Intel	Focs NV	Focs (array) NV
Mitjana ús CPU	~17%	~15%	~14%	~13%
Principal recurs usat	Drivers gràfics	Drivers gràfics	Drivers gràfics	Drivers gràfics
Percentatge	99.5%	99.5%	99.2%	99.5%

El recurs més utilitzat en cadascuna de les execucions és el mateix per a totes, i això es dóna precisament per la tècnica *glAccum*, la qual genera una estela en els vèrtexs en pantalla i, aquest fet, carrega molt l'ús de la visualització, de manera que es generen aquests resultats tan curiosos.

Per últim, s'ha realitzat un anàlisis utilitzant les mateixes condicions però variant el número de partícules per tal d'observar si això influeix en el rendiment. S'ha usat la font sense arrays de vèrtexs, i s'ha executat amb 5.000.000 de partícules. Els resultats es mostren en la figura 9.

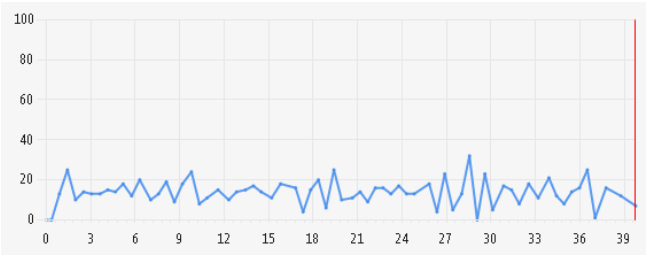


Fig. 9 Utilització de CPU per una execució de la font amb 5 milions de partícules.

Es pot observar que la CPU té més moviment, doncs el gràfic presenta més canvis en el pas del temps. Però, executar tantes partícules no afecta en el rendiment, simplement provoca una execució més lenta. Utilitzant el sistema amb array de vèrtex s'obté una execució més ràpida, però el percentatge de consum de CPU és lleugerament inferior.

2.7 Entorns 3D

S'han implementat els dos sistemes de partícules utilitzant els dos tipus d'algorismes i resta realitzar la importació en entorns 3D. Per a fer-ho, primer s'ha de crear una escena 3D, i per a tal fi s'utilitzaran un torus com a objecte del que fer sortir la font i dos cubs per tal de donar més vida a l'escena.

Gràcies a la parametrització es pot cridar el sistema de partícules indicant el punt d'origen, el que permet situarlo en qualsevol punt de l'escena. Per a crear una escena s'utilitzaran dues fonts emergents des de l'objecte torus, on el resultat que produeix és el següent:

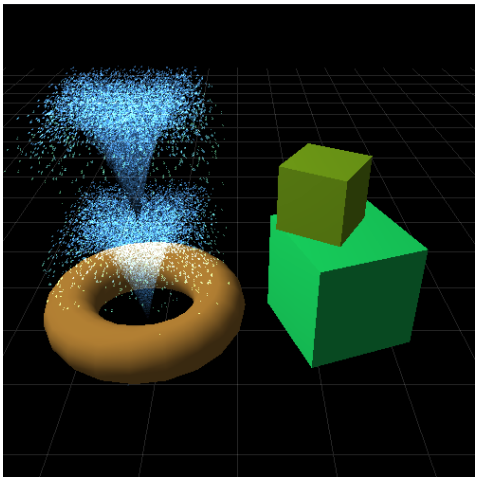


Fig. 10 Escena 3D amb el sistema de partícules de la font.

Els objectes de l'escena s'han creat a partir de les funcions que proveeix OpenGL *glutSolidCube* i *glutSolidTorus*.

A més a més, s'ha aplicat la il·luminació de Gouraud amb shaders per donar un efecte més realista i interessant.

3 CONCLUSIONS

Com a conclusions del projecte realitzat es poden destacar els següents punts:

- Els sistemes de partícules són recursos amb molta potència i de molt ús, i en el projecte se'n han desenvolupat dos de diferents per analitzar el seu comportament a nivell de rendiment utilitzant vàries formes de implementació (amb arrays de vèrtexs i sense).

- S'han realitzat comparacions entre els dos sistemes amb els diferents algorismes (sense shaders) per estudiar les seves diferències respecte el rendiment de CPU.

- La complexitat dels algorismes és elevada però disposen d'una enorme flexibilitat per desenvolupar tot tipus de comportaments (ja siguin simulacions de líquids, fogueres, explosions, etc.).

- Els sistemes de partícules són compatibles amb un potent recurs, els shaders, els quals produeixen efectes visuals més que interessants. La seva complexitat de programació, però, també és alta.

- La capacitat de parametrització i encapsulament d'aquests algorismes obra la porta a tot tipus de pràctiques en diferents àmbits, des de videojocs fins a cinema. Són un recurs molt utilitzat en efectes especials.

Després de la realització del projecte, però, se'n podria destacar una sèrie de millores:

- Tractar diferents comportament en els sistemes de partícules, ja que hi ha moltes més opcions apart de les dues implementades.

- Ampliar l'ús dels shaders per funcionar amb el moviment de les partícules, doncs en el projecte s'usen per pintar-les i aplicar il·luminació.

- Combinar les dues propostes anteriors per aconseguir un sistema per importar en una escena i obtenir un resultat més realista i vistós.

AGRAÏMENTS

Durant el desenvolupament del projecte ha hagut estancaments, moments de problemes on no se sabia com seguir. I en aquests moments, els consells i l'ajuda del tutor del projecte, Enric Martí, han estat de grandíssima ajuda. Sempre ha estat disponible per consultar qualsevol dubte o problema que pogués sorgir, i els agraïments van cap a ell. Ha estat un pilar de gran importància per poder realitzar aquest projecte.

BIBLIOGRAFIA

[1] Sabbarton, Richard. "OpenGL Particle Systems using PointSprites, Textures and Blending in Microsoft Visual Studio C++ 2008 Express Edition", document 08.

[2] Wolf, David. "OpenGL 4.0 Shading Language Cookbook", Packt Publishing Ltd., 2011.

[3] Jacobs, Scott. "Game Programming Gems 7", Course Technology, 2008.

[4] Nguyen, Hubert. "GPU Gems 3", Addison-Wesley, 2008.

[5] The Khronos Group. "Open Graphics Library (OpenGL)", http://www.opengl.org.

[6] The Khronos Group. "OpenGL Shading Language (GLSL)", http://www.opengl.org/documentation/glsl.

[7] The Khronos Group. "GLUT – The OpenGL Utility Toolkit", http://www.opengl.org/resources/libraries/glut.

[8] "The OpenGL Extension Wrangler Library", http://www.glew.sourceforge.net/index.html.