Projet de logiciels et matériels graphiques: Système de particules (feu)

*Ecrit et conçu par **Nicolas Lutz***

Table des matières

Introduction....1
Système de particules et mise à jour....1
  Particles....1
  Système de particules....2
  Foncteurs....2
Passage du CPU au GPU....3
  VBO....3
  VAO....4
  Alpha Blending....4
Autres....5
Album photo....5

Introduction

Le document suivant constitue le rapport du projet de logiciels et matériels graphiques du master ISI (Informatique et Sciences de l'Image) au semestre d'Automne de l'année 2015-2016. Il rapporte les étapes significatives dans la conception et l'écriture du projet, codé en langage C++ et en utilisant la library glm.

Un album online d'images gif animées est disponible à la fin, en plus de deux screenshots.

/!\ Il est impossible de configurer le système via fichier externe, et le programme ne supporte pas d'autre effet que le feu. /!

Système de particules et mise à jour

Particles

Les particules constituent l'entité principale du programme. Il s'agit de classes respectant la propriété dite de *standard layout type*, vérifiée dans le constructeur du système de particules grâce à l'assertion suivante :

```
{
    TP_ASSERT(std::is_standard_layout<Particle>::value, "Particle is not a %s\n", "standard layout type");
}
```

On peut alors utiliser la directive pragma pack pour être certain que le compilateur n'alignera pas la classe avec les mots ou la page :

```
#pragma pack(push, 0)
class Particle {...};
#pragma pack(pop)
```

Les particules en elles-mêmes sont composées des membres suivants, classés en deux catégories (attributs glsl, et non-attributs glsl) :

```
//Attributes//
glm::vec3       m_afPosition;       ///< current position of the particle

glm::vec4       m_afColor;

float           m_fRotation;

float           m_fWidth;           ///< current width of the particle
float           m_fHeight;          ///< current height of the particle
float           m_fTtl;            ///< time to live (in float because we need to pass it to the gpu)

//Non attributes: will be ignored by the vao (but still fetched to the vbo!!)//
glm::vec3       m_afVelocity;   ///< used by the functors to work according to the current particle's velocity.
int             m_iElapsedTime; ///< used to deduce the initial time coupling it with the ttl.
```

Le ttl étant en float mais portant un entier, il est limité à 16,777,216 (dernier float de la suite numérique n+1 représentant un entier correct, 16,777,217 n'ayant pas de représentation correcte). (UPDATE : On rajoutera à ces membres un float stockant la distance avec la caméra en world space). Enfin, leur texture est gérée dans main.cpp (placée dans un sampler uniforme du FS).

Système de particules

Les particules sont gérées par un système de particules, dont la fonction update() est appelée à chaque trame. Le système de particules les stocke de la manière suivante:

```
///the particles themselves
std::vector<Particle>   m_particles;
```

Oui, juste un vecteur. Il est trié en fonction de la distance de la particule à la caméra, et surtout du Ttl de la particule ; ainsi, à la fin de l'itération et du tri, l'itérateur revient supprimer progressivement toutes les particules invalides.

Le système de particules s'occupe également de faire la liaison entre les données du CPU et celles du GPU, que nous étudierons plus tard.

Foncteurs

Des foncteurs sont passés **par référence** à la fonction update() du système de particules. Ils sont de 5 catégories, dont le nom est très explicite de leur but:

```
///Interfaces

class IFunc_UpdateParticle {...};

class IFunc_UpdateParticle_Position : public IFunc_UpdateParticle {...};

class IFunc_UpdateParticle_Color : public IFunc_UpdateParticle {...};

class IFunc_UpdateParticle_Rotation : public IFunc_UpdateParticle {...};

class IFunc_UpdateParticle_Size : public IFunc_UpdateParticle {...};

class IFunc_UpdateParticle_Init : public IFunc_UpdateParticle {...};
```

Et sont appelés dans l'ordre sur chaque particule **vivante** (sauf dans le cas des foncteurs de type Init, qui sont utilisés une seule fois à la naissance d'une particule, et servent de pseudo-constructeur aux particules).

Ce système permet de modifier dynamiquement le comportement des particules grâce à des fonctions pré-installés ; si on va plus loin, il est même possible de leur passer des scripts. **Cependant**, il peut y avoir un problème de performances avec cette solution, et des templates de foncteurs peuvent être plus adaptés dans un cas aussi fixe que le notre ; au final, j'ai préféré la modularité. De plus, les foncteurs sont, dans la plupart des cas, suffisamment proches du vecteur en mémoire pour que le cache puisse charger les deux zones en même temps.

Note : j'ai récemment songé que passer une référence sur un vecteur de foncteurs à la fonction update() pourrait être une meilleure idée afin de cumuler plus d'effets.

Exemple de foncteur : convergence du vecteur vitesse vers un vecteur direction

```
Particle& Func_UpdateParticle_Position_Direction_Converge::operator()(Particle& particle)
{
    glm::vec3 currentVelocity(particle.getVelocity());
    //get length and normalize (without wasting time)
    float len=glm::length(currentVelocity);
    currentVelocity/=len;

    //We can now compare our vectors and complete them
    currentVelocity.x+=(currentVelocity.x>afDirection.x ? -fconvergeIntensity : fconvergeIntensity);
    currentVelocity=glm::normalize(currentVelocity);
    currentVelocity.y+=(currentVelocity.y>afDirection.y ? -fconvergeIntensity : fconvergeIntensity);
    currentVelocity=glm::normalize(currentVelocity);
    currentVelocity.z+=(currentVelocity.z>afDirection.z ? -fconvergeIntensity : fconvergeIntensity);
    currentVelocity=glm::normalize(currentVelocity);

    currentVelocity*=len;

    //give the same velocity, except modified with the proper angle
    particle.setPosition(particle.getPosition()
                         + currentVelocity);
    particle.setVelocity(currentVelocity);
    return particle;
}
```

Passage du CPU au GPU

VBO

Les particules évoluent au cours du temps et leur nombre grandit. On utilisera donc GL_STREAM_DRAW pour passer les données au VBO. Les particules sont contigus dans le vecteur, la propriété de *standard layout type* des particules garantit qu'elles n'ont pas de *virtual table* et la directive *pragma pack* certifie qu'elles ne sont pas alignées sur les mots ou les pages, on peut donc passer le vecteur entier et cru dans **un seul et unique VBO.**

Il y a un seul VBO, initialisé de la façon suivante :

```
glGenBuffers( 1, &m_iVBOParticle );

// Binds the Buffer, to say "we will work on this one from now"
glBindBuffer( GL_ARRAY_BUFFER, m_iVBOParticle );
{
    glBufferData( GL_ARRAY_BUFFER, m_maxSize * sizeof(Particle), NULL, GL_STREAM_DRAW );
}
// Unbinds the Buffer, we are done working on it !
glBindBuffer( GL_ARRAY_BUFFER, 0 );
```

m_maxSize est une taille pseudo-maximale qui double lorsque le vecteur est suffisamment grand. Elle permet de créer du buffer orphaning (buffer re-specification) et d'améliorer les performances.

Le VBO est donc mis à jour de la manière suivante :

```
// VBO UPDATE: our particle system isn't static and both its size and content change
glBindBuffer(GL_ARRAY_BUFFER, m_iVBOParticle);
{
    // Buffer orphaning: enhances performances
    glBufferData(GL_ARRAY_BUFFER, m_maxSize*sizeof(Particle), NULL, GL_STREAM_DRAW);

    //Data filling using glBufferSubData
    glBufferSubData(GL_ARRAY_BUFFER, 0, m_particles.size()*sizeof(Particle), &m_particles[0]);
}
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Toujours en utilisant le buffer orhphaning.

VAO

Il n'y a qu'un seul VAO. Il fait la liaison entre les locations et la classe Particle de la façon suivante :

```
    glBindVertexArray( m_iVAO );
    {
        if( 0 != m_iVBOParticle )
        {
            glBindBuffer( GL_ARRAY_BUFFER, m_iVBOParticle );
/*position*/glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Particle), NULL); //takes 12 bytes
            glEnableVertexAttribArray(0);
/*color*/   glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, sizeof(Particle), (void*)12); //+16 bytes
            glEnableVertexAttribArray(1);
/*rotation*/glVertexAttribPointer(2, 1, GL_FLOAT, GL_FALSE, sizeof(Particle), (void*)28); //+4 bytes
            glEnableVertexAttribArray(2);
/*width*/   glVertexAttribPointer(3, 1, GL_FLOAT, GL_FALSE, sizeof(Particle), (void*)32); //+4 bytes
            glEnableVertexAttribArray(3);
/*height*/  glVertexAttribPointer(4, 1, GL_FLOAT, GL_FALSE, sizeof(Particle), (void*)36); //+4 bytes
            glEnableVertexAttribArray(4);
/*ttl*/     glVertexAttribPointer(5, 1, GL_FLOAT, GL_FALSE, sizeof(Particle), (void*)40); //+4 bytes
            glEnableVertexAttribArray(5);
            glBindBuffer( GL_ARRAY_BUFFER, 0 );
        }
    }
     // Stops using the VAO
    glBindVertexArray( 0 );
```

En utilisant les strides et les offsets de façon intensive. Les offsets sont mis en dur. Les membres non-attributs sont ignorés par le VAO (mais, encore une fois, ils sont bel et bien dans le VBO)

Alpha Blending

La nature des particules nous oblige à utiliser de l'*alpha blending* pour obtenir un rendu plus réaliste. C'est une technique coûteuse, car elle requiert de trier les triangles du plus éloigné au plus proche pour pouvoir rendre les plus proches en dernier. Le problème est cependant facilité par la nature d'une particule <=> un point, donc il suffit de trier les points à la fin de l'itération :

```
//Sort the particles in function of their distance to the camera
std::sort(m_particles.begin(), m_particles.end(), m_funcCompare);
                                                  Func_ParticleCompare
```

l'opérateur de comparaison du foncteur Func_ParticleCompare est défini de la façon suivante :

```cpp
bool operator()(const Particle& object, const Particle& other)
{
    if(object.getTtl()<=0)
    {
        if(other.getTtl()>0)
            return false;
    }
    return object.getDistanceToCamera()>other.getDistanceToCamera();
}
```

Notez que le Ttl a une priorité plus importante. C'est pour pouvoir revenir dans le vecteur en supprimant les particules mortes, comme expliqué au-dessus.

Enfin, la distance à la caméra est mise à jour de la façon suivante à chaque trame:

```cpp
void Particle::updateDistanceToCamera(const glm::mat4& objectToWorld, const glm::vec3& cameraWorldPos)
{
    glm::vec3 worldPosition=glm::vec3(objectToWorld*glm::vec4(m_afPosition,1.0f));
    m_fdistanceCamera=glm::distance(worldPosition, cameraWorldPos);
}
```

où objectToWorld est la matrice qui permet de passer de l'espace Object du système de particule à l'espace World, et cameraWorldPos la position de la caméra déjà dans l'espace World.

Autres

- Les commandes sont identiques que pour les précédents TP (sans la touche P), donc on peut désactiver l'utilisation de l'alpha blending avec B et celle des couleurs avec C.
- De plus, il est possible d'éteindre (et de rallumer) le feu avec la touche S, ce qui a également pour effet de disperser les particules restantes.
- La création du programme graphique est strictement identique à celle du dernier TP.
- Le c++11 est juste là pour valider la classe Particle, il est donc possible de le retirer, si nécessaire, en supprimant la vérification dans le constructeur de ParticleSystem, le header <type_traits> de particlesystem.h et en enlevant la ligne CONFIG+=c++11 du fichier .pro.
- Ne pas oublier de rajouter make install dans qt.

Album photo

Gifs enregistrés pendant le développement !

Première version potable, random spread dans la mise à jour: https://i.gyazo.com/5f181a615c3bb76364d3f1615e6de21a.gif

Seconde version, avec convergence vers le haut et sans accélération: https://i.gyazo.com/c7348a562d8607504e002e5072b21257.gif

Avec des couleurs aléatoires: https://i.gyazo.com/f487072fcb660583c613d0ea3c4794bc.gif

Troisième version, en réalisant que mon alpha blending était faux (particules mal triées) et en diminuant progressivement l'alpha :

https://i.gyazo.com/39c906227df9fac92091dbb18abe3a29.gif

Quatrième version, en réduisant progressivement la taille des particules à un seuil de 30% de vie, avec un angle plus sympa et une rotation pour bien voir qu'on est en 3D:
https://i.gyazo.com/f0243eb070f49489ef97a599a22f240c.gif

Cinquième version, cette fois en rajoutant en plus une rotation aux particules : https://i.gyazo.com/b22fd11fdd3b0bddf1a490358a0182b2.gif

En appuyant sur la touche S puis en maintenant espace : https://gyazo.com/00fe7c5bdaf4b2823c9b0e8bebaca01b