Initiation à la GPGPU

Introduction

David Odin

Forma3Dev pour CPE–Lyon

2016

Notes

Qu'est-ce que la GPGPU ?

- **G**eneral **P**rograming on **G**raphic **P**rocessing **U**nits.
- Utilisation de la puissance des cartes graphiques pour autre chose qu'afficher des triangles.

Notes

Programmation parallèle

- Idée assez vieille, au moins 30 ans (transputers)
- Implémenté via MMX, SSE sur les processeurs *x86* depuis des années.
- Altivec sur PowerPC, Paired Single sur Wii, NEON sur ARM, etc.
- **S**ingle **I**nstruction on **M**ultiple **D**ata.

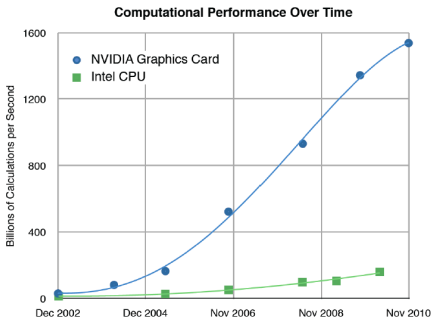
Notes

POURQUOI SUR GPU ?

- Les GPU sont SIMD par nature (*fragment shader*).
- Shaders unifiés.
- GPGPU commence à devenir mature (standardisation).
- CPU = 24 cœurs d'exécutions max.
- GPU = 3200 cœurs pour la AMD Radeon HD 5970 (comparable pour NVidia).

Notes

COMPARAISON CPU / GPU



Source: Cyprien Adnet.

Notes

LIMITATIONS

CPU $\neq$ GPU.

- Beaucoup de données en même temps, oui, mais un seul programme à la fois.
- Impossibilité d'écrire et de lire les mêmes données en même temps (sauf toute dernière génération, mais demande une synchro)
- Pas applicable à tous les algorithmes.

Notes

TRAITEMENTS D'IMAGE SIMPLES

- Traitements les plus simples possibles
- Noir & blanc
- Sépia
- Ajustement de contraste
- Ajustement de luminosité
- Seuillage

Notes

CONVOLUTIONS

- Flou
- Érosion / Dilatation
- Post-effects :
  - DoF (profondeur de champ),
  - HDR (simulation de conditions de lumière extrèmes)

Notes

RENDU VOLUMIQUE

La GPGPU permet de faire du rendu volumique :

- Raycasting
- Raytracing
- Radiosité
- Raymarching
- Imagerie médicale

Notes

AUTRES

Calculs dans un système de rendu

- Création de terrains complexes
- Système de particules
- Physiques (mass-spring, rigid body, etc.)
- Calculs de collisions
- Skinning

Notes

PLUS AMBITIEUX

- Calculs de pliage de protéines
- Équation aux dérivées partielles
- Calculs météo
- Mécanique des fluides
- Cryptage / décryptage
- Détection de virus

Notes

MISE EN ŒUVRE

- OpenGL 2.1 : Fragment shader et FBO.
- OpenGL 3.0 : Transform feedback.
- OpenCL : Computing library.
- OpenGL 4.3 : Compute shader.

Notes

CONSTAT OPENGL 2.X

- Donnée de base (sortie) : pixel/fragment.
- Programme identique pour beaucoup de données : *Fragment shader*.
- Données d'entrées :
  - Variables *uniform*,
  - Variables *varying*,
  - Textures 1D, 2D, 3D, etc.

Notes

EXEMPLE : IMPLÉMENTATION DU FLOU

- À partir d'une image stockée dans une texture, on affiche un *quad* de la même taille à l'écran avec le *fragment shader* suivant :

```
1  uniform sampler2D unite_texture;
2  uniform vec2      taille_texture;
3
4  in vec2 v_tex_coord;
5
6  out vec4 out_color;
7
8  void main(void)
9  {
10   vec2 offset = vec2(1.0, 1.0) / taille_texture;
11
12   vec4 color  = texture(unite_texture, v_tex_coord + offset * vec2(-1.0, -1.0));
13        color += texture(unite_texture, v_tex_coord + offset * vec2(-1.0,  0.0));
14        color += texture(unite_texture, v_tex_coord + offset * vec2(-1.0,  1.0));
15        color += texture(unite_texture, v_tex_coord + offset * vec2( 0.0, -1.0));
16        color += texture(unite_texture, v_tex_coord + offset * vec2( 0.0,  0.0));
17        color += texture(unite_texture, v_tex_coord + offset * vec2( 0.0,  1.0));
18        color += texture(unite_texture, v_tex_coord + offset * vec2( 1.0, -1.0));
19        color += texture(unite_texture, v_tex_coord + offset * vec2( 1.0,  0.0));
20        color += texture(unite_texture, v_tex_coord + offset * vec2( 1.0,  1.0));
21
22   out_color = color / 9.0;
23 }
```


Notes

FBO

- Espace de rendu caché
- Attaché à une ou plusieurs textures
- Peut contenir 4 textures : ajoute des données de sorties !
- Peut contenir une ”texture de profondeur”
- Peut contenir des textures de flottants
- Se comporte comme le *framebuffer* ”normal”
- Utilisation en ”Ping-Pong”

Notes

Implémentation des FBO

- Création et activation :

```
GLuint fbo;
glGenFramebuffers(1, &fbo);
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
```

- Attachement d'une texture :

```
GLuint fbo_texture;
glGenTextures(1, &fbo_texture);
glBindTexture(GL_TEXTURE_2D, fbo_texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 800,600, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, NULL);
glFramebufferTexture2D(GL_FRAMEBUFFER,
                       GL_COLOR_ATTACHMENT0,
                       GL_TEXTURE_2D, fbo_texture, 0);
```

- On peut également ajouter d'autres buffers (couleur, profondeur ou stencil)
- Réactivation du framebuffer par défaut (la fenêtre) :

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```


Notes

Utilisation classique des FBOs

Rendu en deux passes :

```
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
...
glDrawElements(GL_TRIANGLES, 24,
               GL_UNSIGNED_SHORT, 0);
```

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glBindTexture(GL_TEXTURE_2D, fbo_texture);
...
glDrawElements(GL_TRIANGLES, 6,
               GL_UNSIGNED_SHORT, 0);
```


Notes

Utilisation "GPGPU" en ping-pong

- Utilisation de 2 FBO
- On utilise la texture du FBO 1 quand on écrit dans le FBO 2
- Et réciproquement
- Le résultat est utilisé pour le rendu

Notes

Implémentation du ping-pong

- Création des deux FBO avec chacun leur texture.

```
frame++;
if (frame & 1) // Si frame est impair
{
  i = 1; j = 0;
} else {
  i = 0; j = 1;
}
```

```
glBindTexture(GL_TEXTURE_2D, fbo_texture[i]);
glBindFramebuffer(GL_FRAMEBUFFER, fbo[j]);
...
glDraw...
```

```
glBindTexture(GL_TEXTURE_2D, fbo_texture[j]);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
...
glDraw...
```


Notes

Transform Feedback

- Disponible depuis OpenGL 3
- Récupère la sortie du vertex shader
- (Ou du geometry shader ou du tesselation shader)
- Sortie dans un VBO
- Variables d'entrées : attribute, uniform, textures
- Désactivation du dessin :

```
glEnable(GL_RASTERIZER_DISCARD);
```


Notes

Transform Feedback, exemple de shader

Animation de particules.

- Seul le Vertex Shader est nécessaire, par exemple :

```
uniform vec3 acceleration = vec3(0.0, -9.8, 0.0);
uniform mat4 projection;

in vec3 vitesse;
in vec3 position;
out float v_vitesse;
out float v_position;

void main()
{
  v_vitesse = vitesse + acceleration;
  v_position = position + acceleration;
  gl_Position = projection * v_position;
}
```


Notes

TRANSFORM FEEDBACK, COMPILATION

- Compilation du programme / shader :
  On indique les varying à capturer **avant** l'édition de liens :

```
1  GLuint shader = glCreateShader(GL_VERTEX_SHADER);
2  glShaderSource(shader, 1, &vertexShaderSrc, nullptr);
3  glCompileShader(shader);
4
5  GLuint program = glCreateProgram();
6  glAttachShader(program, shader);
7  glBindAttribLocation(program, 0, "position");
8  glBindAttribLocation(program, 1, "vitesse");
9
10 const GLchar *varyings[] = { "v_position", "v_vitesse" };
11 glTransformFeedbackVaryings(program, 2, varyings,
12                             GL_INTERLEAVED_ATTRIBS);
13 glLinkProgram(program);
14 glUseProgram(program);
```


Notes

TRANSFORM FEEDBACK, UTILISATION

- Création d'un buffer, de manière classique

```
1 GLuint vbo_tf;
2 vec3 data[2 * NB_PARTICLES];
3 glGenBuffers(1, &vbo_tf);
4 glBindBuffer(GL_ARRAY_BUFFER, vbo_tf);
5 glBufferData(GL_ARRAY_BUFFER, sizeof data, nullptr, GL_STREAM_READ);
```

- Utilisation : enregistrement des données

```
1 glEnable(GL_RASTERIZER_DISCARD);
2 glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, 0, tbo);
3 glBeginTransformFeedback(GL_POINTS);
4 glDraw...(GL_POINTS,...);
5 glEndTransformFeedback();
6 glDisable(GL_RASTERIZER_DISCARD);
```

- Récupération des données

```
1 glGetBufferSubData(GL_TRANSFORM_FEEDBACK_BUFFER,0,sizeof data,data);
```

  Ou par l'utilisation du vbo directement par un autre glDraw

Notes

OPENCL

- Standard ouvert pour le calcul sur les GPU (OpenGL 3+)
- Conçu pour s'exécuter sur GPU, CPU, SPU, DSP, etc.
- Utilisable sur GPU "compatible OpenGL 3.x+"
- Langage "parallèle" semblable à GLSL ou au C++
- Gère la mémoire vidéo de manière transparente pour le programmeur
- Gère les cœurs d'exécutions du GPU

Notes

COMPUTE SHADERS

- Depuis OpenGL 4.3
- Utilise GLSL
- S'utilise un peu comme avec les Transform Feedback (buffer d'entrée et buffer de sortie)
- Les appels à glDraw() sont remplacés par glDispatchCompute()
- Même contrôle fin de la mémoire et des threads que pour OpenCL

Notes

Notes

Notes