Raytracing en GPU

Por Matías Haeussler

Departamento de Ciencias de la Computación
UNIVERSIDAD DE CHILE

CC7515 - Computación en GPU

12/12/2017

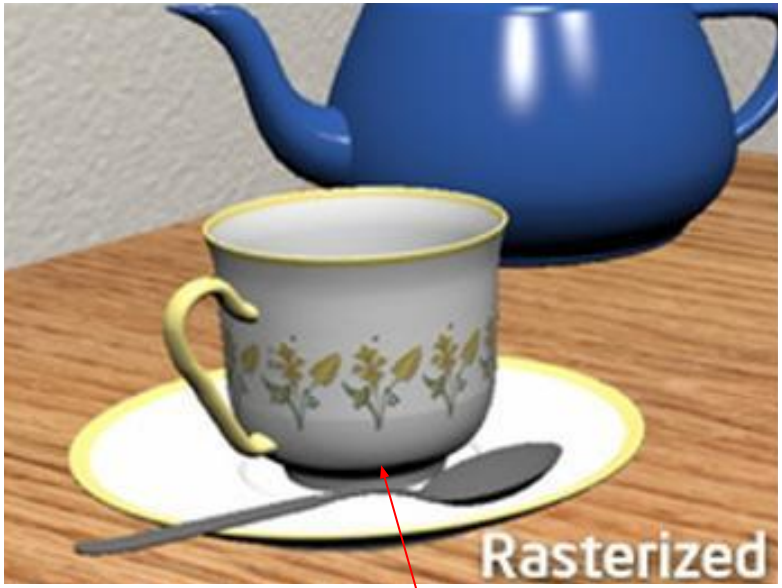
Rasterization vs Raytracing:

Imágenes

Rasterization vs Raytracing:

Imágenes

Materiales opacos



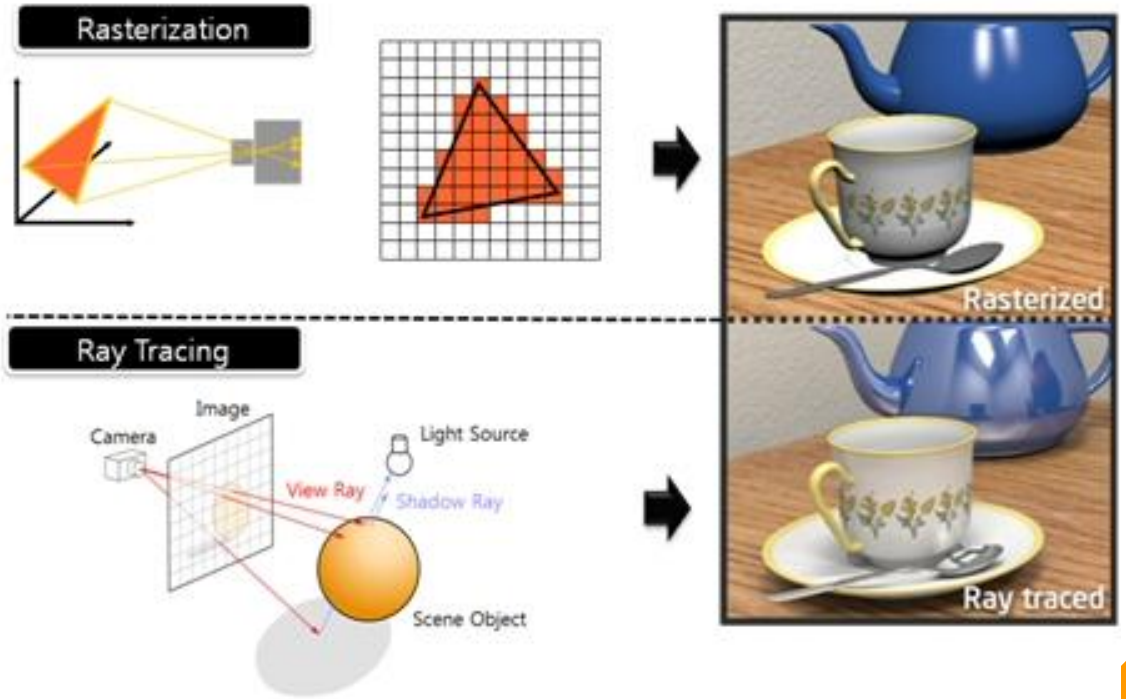
Sombra no natural

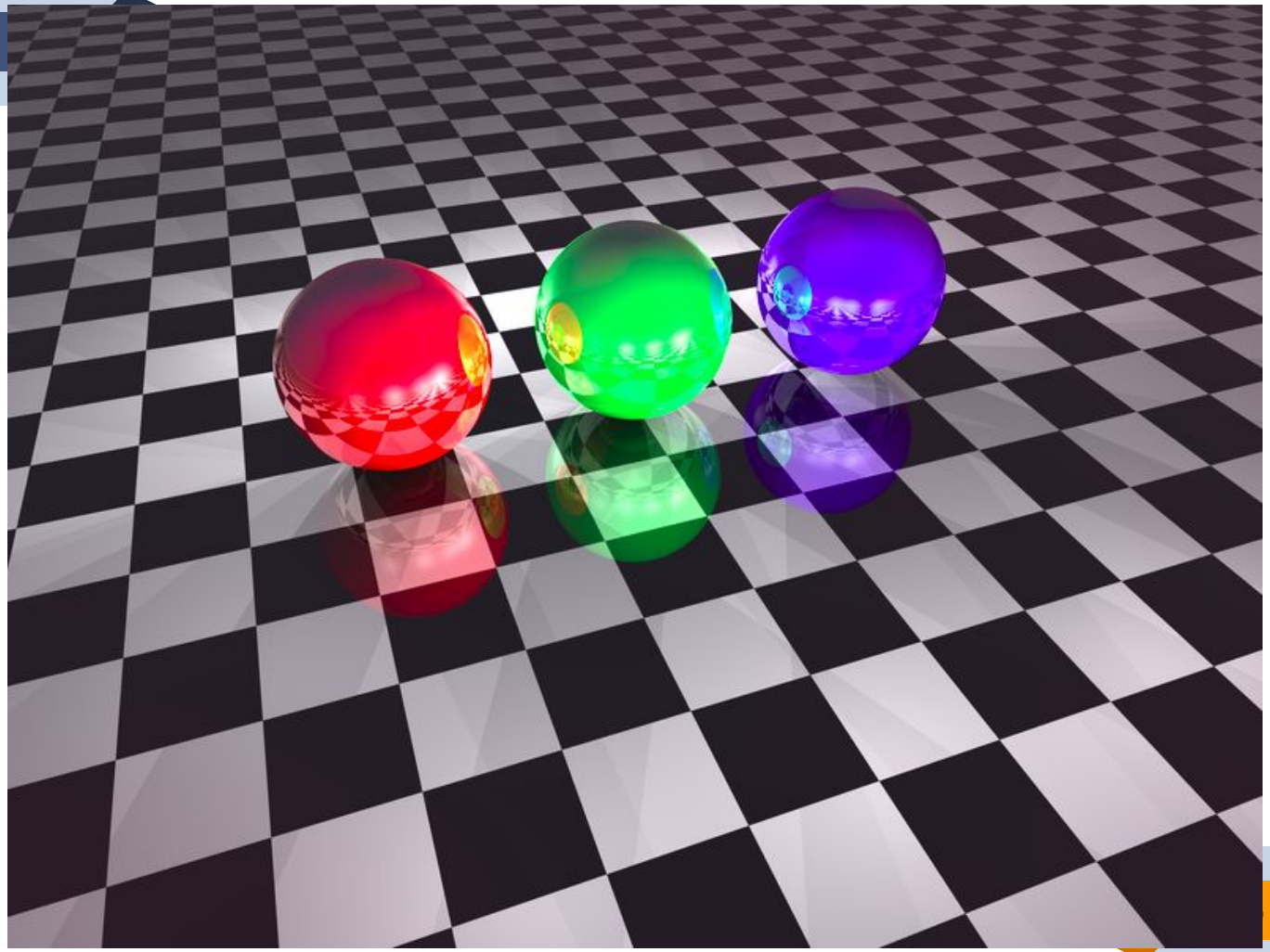
Reflección

Sombra realista

Rasterization vs Raytracing:

Funcionamiento






OpenGL® ES ray tracing
PowerVR
by Imagination

CPU:

Modelo en C++

Class:
- Camera
- Ray
- Object
- Material
- Light

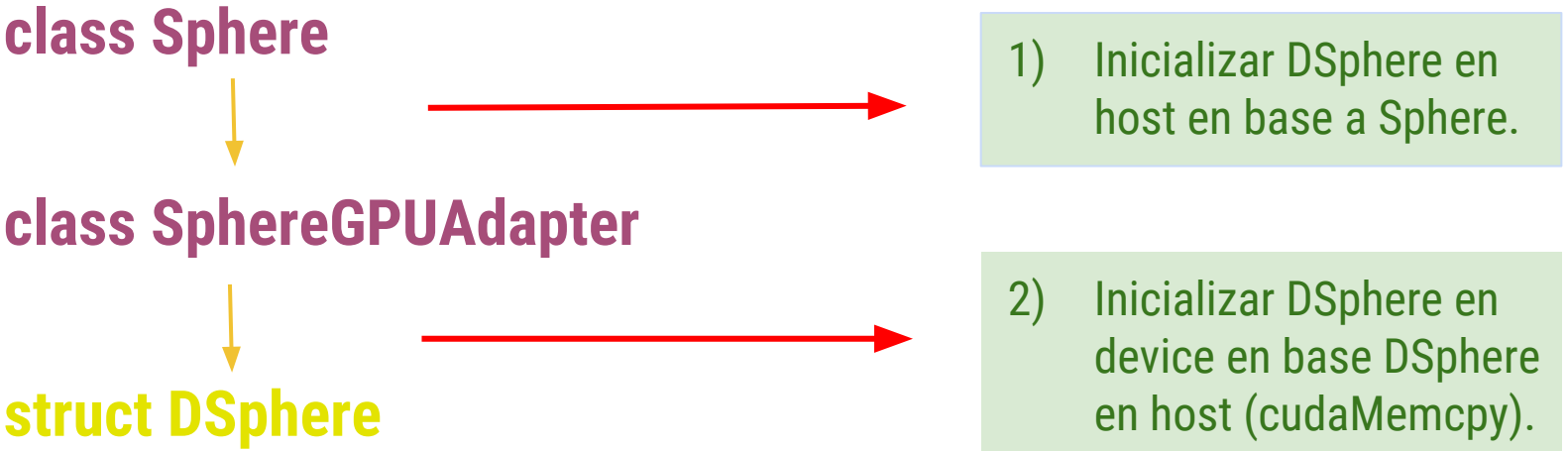
Light:
- Punctual
- Directional
- Spotlight
- Ambiental

Material:
- Lambert
- Phong
- Reflective
- Dielectric
- Texture

Object:
- Sphere
- Mesh

De CPU a GPU:

Adapter Pattern



```
842
843 SphereGPUAdapter::SphereGPUAdapter(Sphere *s) {
844
845   // Init GPU point & assosiated pointers
846
847   int nMats = s->getMaterials().size();
848   h_s = (DSphere*)malloc(sizeof(DSphere));
849   for (int i=0; i<3; i++) h_s->c[i] = s->getCenter()[i];
850   h_s->r = s->getRadius();
851   h_s->nMats = nMats;
852
853   cudaMalloc((void **)&d_s, sizeof(DSphere));
854   cudaMalloc((void **)&d_mats, nMats*sizeof(DMaterial*));
855   htod_mats = (DMaterial**)malloc(nMats*sizeof(DMaterial*));
856   for (int i=0; i<nMats; i++)
857     htod_mats[i] = s->getMaterials()[i]->buildDMaterial();
858
859
860   // Copy host pointers to GPU
861
862   cudaMemcpy(d_s, h_s, sizeof(DSphere),
863       cudaMemcpyHostToDevice);
864   cudaMemcpy(d_mats, htod_mats, nMats*sizeof(DMaterial*),
865       cudaMemcpyHostToDevice);
866   iniMats<<<1,1>>>(d_s, d_mats);
867 }
868
```

De CPU a GPU:

Memoria y recursión

No utilizar memoria dinámica dentro del algoritmo !!

1) **10x SpeedUp**
2) **Arreglos con tamaño fijo**

No usar recursión !!
(al reflejar o refractar rayos)

1) **Utilizar stacks**
2) **(Tamaño máximo fijo)**

```
388
389 __device__ void intersectRay(DStack *stack, DRay *ray,
390     DSpheres *sphs, DCamera *cam, const DLights *ls) {
391
392     for (int is=0; is<sphs->nSpheres; is++)
393       intersectSphere(sphs->s[is],sphs,ray,cam);
394
395     if (ray->s) {
396       for (int im=0; im<ray->s->nMats; im++) {
397         colorate(stack,ray->s->mats[im],sphs,ls,cam,ray);
398       }
399     } else copyVec(ray->col, cam->bg_col);
400 }
401
402
```

Demo

De CPU a GPU:

Resultados

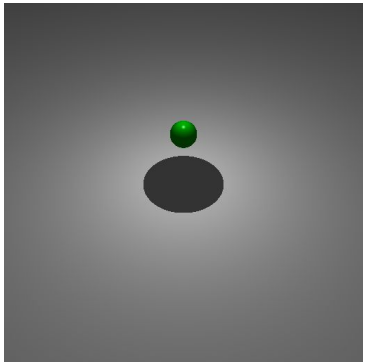


Imagen	CPU (s)	GPU (s)	SpeedUp
128x128	0.25	0.0004	625
256x256	0.91	0.0007	1300
512x512	4.43	0.0021	2109

De CPU a GPU:

Resultados

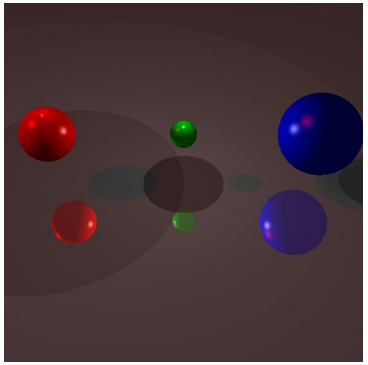


Imagen	CPU (s)	GPU (s)	SpeedUp
128x128	0.68	0.0006	1133
256x256	2.63	0.0014	1879
512x512	10.45	0.0046	2271

De CPU a GPU:

Resultados

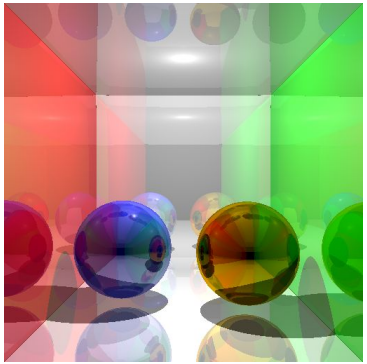


Imagen	CPU (s)	GPU (s)	SpeedUp
128x128	2.00	0.0010	2000
256x256	8.01	0.0029	2762
512x512	32.59	0.0098	3326

Demo

```
11 unsigned char* getMappedBuffer(unsigned short width,
12   unsigned short height) {
13
14   glfwInit();
15   window = glfwCreateWindow(width, height, "Window", NULL, NULL);
16   glfwMakeContextCurrent(window);
17   glewExperimental = GL_TRUE; glewInit();
18
19   glGenBuffers(1, &gl_pixelBufferID);
20   glBindBuffer(GL_PIXEL_UNPACK_BUFFER, gl_pixelBufferID);
21   glBufferData(GL_PIXEL_UNPACK_BUFFER, width*height*4, NULL,
22       GL_DYNAMIC_COPY);
23
24   glEnable(GL_TEXTURE_2D);
25   glGenBuffers(1, &gl_textureID);
26   glBindTexture(GL_TEXTURE_2D, gl_textureID);
27   glTexParameteri(GL_TEXTURE_2D,
28       GL_TEXTURE_MAG_FILTER, GL_NEAREST);
29   glTexParameteri(GL_TEXTURE_2D,
30       GL_TEXTURE_MIN_FILTER, GL_NEAREST);
31   glTexParameteri(GL_TEXTURE_2D,
32       GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
33   glTexParameteri(GL_TEXTURE_2D,
34       GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
35   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, width, height, 0,
36       GL_RGBA, GL_UNSIGNED_BYTE, NULL);
37
38   cudaGLRegisterBufferObject(gl_pixelBufferID);
39   unsigned char *d_textureBufferData;
40   cudaGLMapBufferObject((void**)&d_textureBufferData,
41       gl_pixelBufferID);
42   return d_textureBufferData;
43 }
44
```

```
44
45 void write_image(unsigned short width, unsigned short height){
46
47   glTexSubImage2D(GL_TEXTURE_2D, 0,0,0, width, height,
48       GL_RGBA, GL_UNSIGNED_BYTE, NULL);
49
50   glClear(GL_COLOR_BUFFER_BIT);
51   glBegin(GL_QUADS);
52     glTexCoord2f(0.0, 0.0);
53     glVertex2f(-1.0, -1.0);
54     glTexCoord2f(0.0, 1.0);
55     glVertex2f(-1.0,  1.0);
56     glTexCoord2f(1.0, 1.0);
57     glVertex2f( 1.0,  1.0);
58     glTexCoord2f(1.0, 0.0);
59     glVertex2f( 1.0, -1.0);
60   glEnd();
61
62   do {
63     glfwSwapBuffers(window);
64     glfwPollEvents();
65
66   } while (glfwGetKey(window, GLFW_KEY_ESCAPE) != GLFW_PRESS &&
67       glfwWindowShouldClose(window) == 0);
68
69   cudaGLUnregisterBufferObject(gl_pixelBufferID);
70   cudaGLUnmapBufferObject(gl_pixelBufferID);
71   glDeleteBuffers(1, &gl_pixelBufferID);
72   glDeleteTextures(1, &gl_textureID);
73   glfwTerminate();
74 }
```

De CPU a GPU:

OctTree

class OctTree

class OctTreeGPUAdapter

struct DOctTree

No usar recursión !!
(al profundizar en el árbol)

1) Utilizar stack para nodos
2) (Tamaño máximo fijo)

```
328
329 __device__ void intersectRay(DRay *ray, DSpheres *sphs,
330     DCamera *cam) {
331
332   DOctTree oct_s[20], *oct; DOctStack *octStack, octStack_s;
333   octStack = &octStack_s; int child;
334   for (int i=0; i<20; i++) octStack->oct[i] = &oct_s[i];
335   initOctStack(octStack);
336   if (ray->oct) {
337     octStackPush(octStack, ray->oct, 0);
338   }
339
340   while (octStack->size) {
341
342     child = topChild(octStack);
343     oct = octStackPop(octStack);
344     if (oct->type == LEAVE) {
345       intersectChildren(oct, ray, cam);
346     } else if (child < 8) {
347       octStackPush(octStack, oct, child+1);
348       intersectOctNode(oct->child[child], ray, cam, octStack);
349     } else continue;
350   }
351
352   for (int is=0; is<sphs->nSpheres; is++)
353     intersectSphere(sphs->s[is],sphs,ray,cam);
354 }
355
```

De CPU a GPU:

Resultados

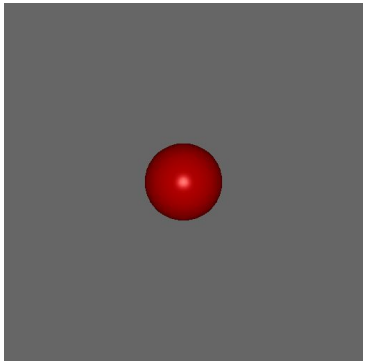


Imagen	CPU (s)	GPU (s)	SpeedUp
128x128	14.80	0.63	23.49
256x256	59.42	1.83	32.47
512x512	233.65	6.45	36.22

Construcción OctTree: 1.13 seg

De CPU a GPU:

Resultados

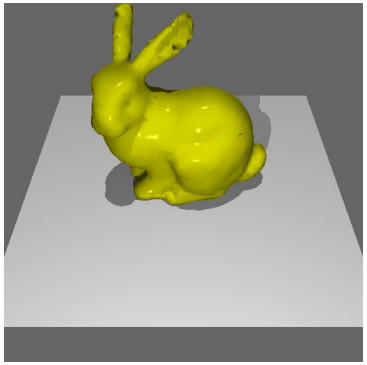


Imagen	CPU (s)	GPU (s)	SpeedUp
128x128	26.22	1.57	16.70
256x256	106.76	4.58	23.31
512x512	427.20	17.87	23.91

Construcción OctTree: 4.57 seg

De CPU a GPU:

Resultados

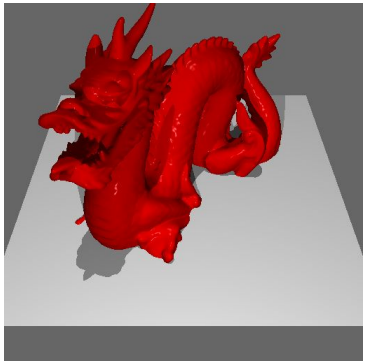


Imagen	CPU (s)	GPU (s)	SpeedUp
128x128	68.95	5.84	11.81
256x256	273.72	16.44	16.65
512x512	1159.60	54.31	21.35

Construcción OctTree: 189.44 seg

Conclusiones:

Comentarios

1) Muy buena interacción entre CUDA y OpenGL
2) Prinf, kernel único y cuda-gdb (PREEMPTIVE)
3) Probar texturas y hacer profiling para el OctTree
4) Provar con varias GPUs

Referencias

1) Técnicas Avanzadas de Rendering:
2) What Every CUDA Programmer Should Know About OpenGL: http://www.nvidia.com/content/gtc/documents/1055_gtc09.pdf
3) CC7615Malloc in CUDA kernel? https://stackoverflow.com/questions/9806299/
4) Best way of traversing an OctTree in CUDA: https://devtalk.nvidia.com/default/topic/409587/