HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

Titel der Arbeit auf Deutsch

English Title

Jeremias Boos

Hausarbeit zur Vorlesung Realtime Rendering

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, 10. März 2016

Kurzfassung

Dies ist eine eines Verfahrens zur Berechnung von 2D Schatten. Das Hier vorgestellte verfahren arbeitet hauptsächlich auf der GPU mit Framebuffern und Fragment Shadern.

Inhaltsverzeichnis

1

Motivation

Mich hat es schon immer interessiert wie Schatten in 2D Spielen umgesetzt. Dazu habe ich zwei verfahren gefunden. Das eine basiert darauf eine Physikengine zu nutzten und dann mit Hilfe von Raycasts einen “Lichtmesh“ zu erzeugen. Das andere Verfahren nutzt die Grafikkarte und berechnet die Schatten im Fragmentshader auf der GPU. Da sich der Prozess gut parallelisieren lässt und Grafikkarten immer stärker werden habe ich mich für die zweiter Variante entschieden.

2

Verfahren

Das Verfahren besteht grundsätzlich aus 3 Schritten.

2.1 Oclussionmap

Zur Licht Berechnung wird zuerst eine Occlusionmap erstellt 2.2. Diese Textur Bildet ab was das licht vom Level sieht.2.1

Abb. 2.1. Testgrafik

2.2 Sample Distanz

Die Oclusionmap wird zur besseren Parallelisierbarkeit in Polarkoordinaten gesampelt.2.3 Nun wird einfach die Distanz für jedes X gemessen bis es auf ein blockierenden Pixel stößt. 2.4 Die gemessene Distanz wird in einer 1D Textur zur weiter Verarbeitung Gespeichert 2.5.

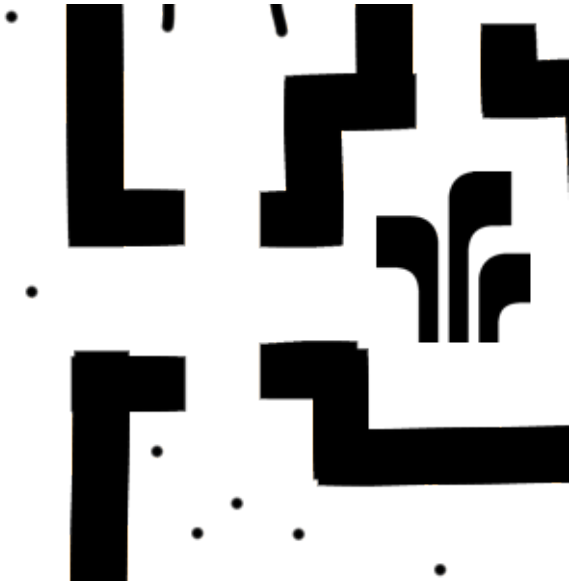

Abb. 2.2. Oclussionmap

Abb. 2.3. Oclussionmap in polar Koordinaten

2.3 Render Licht

Im letzten Schritt wird ein Sprite in der Größe des lichtes gerendert. Durch Verwendung einer Stepfunktion wird die Distanz zum mittel Punkt mit der maximal Entfernung die in der Textur gespeichert wurde verglichen. Anschließend muss das Sprite nur noch additiv Geblendet gerendert werden.2.7

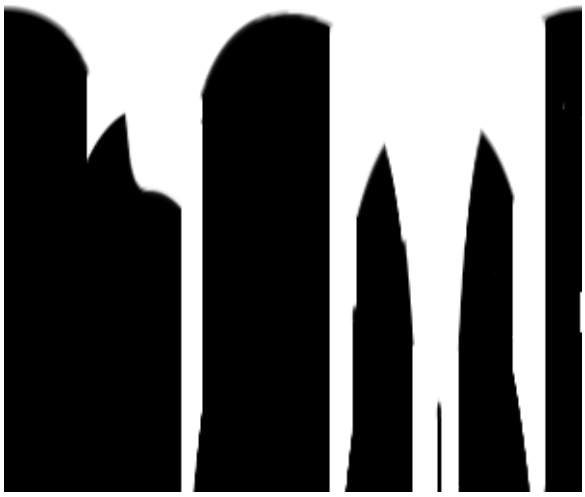

Abb. 2.4. Symbolische Shadowmap in polar Koordinaten

Abb. 2.5. Gesampelte Distanzdaten.

Abb. 2.6. Shadowmap gerendert

Abb. 2.7. Level mit einem Licht

3

Implementierung

Das Verfahren wurde im Rahmen des Projektes in einen OpenGl C++ Programm umgesetzt. Um die 2D Schatten zu implementieren müssen zwingend Framebuffer benutzt werden. Den ohne können die entsprechenden Texturen nicht erstellt werden die zu Berechnung der Schatten benötigt werden.

Abb. 3.1. Programm beim nach dem Start.

Die Schritte die hier beschrieben werden müssen für jedes Licht durchgeführt werden.

3.1 Oclussionmap

Für die Oclussionmap wird eine Textur angelegt die der Größe des Lichtes entspricht.

```
1  // The texture we're going to render to
2  GLuint oclusion_texture;
3  glGenTextures(1, &oclusion_texture);
```

```
4
5  // "Bind" the newly created texture :
6  // all future texture functions will modify this texture
7  glBindTexture(GL_TEXTURE_2D, oclusion_texture);
8
9  // Give an empty image to OpenGL ( the last "0" )
10 glTexImage2D(GL_TEXTURE_2D, 0,GL_RGBA8,
11         ligthsize, ligthsize, 0,
12         GL_RGBA, GL_UNSIGNED_BYTE, 0);
```

Diese wird nun an ein FBO gebunden.

```
1  // Set "oclusion_texture" as our colour attachement #0
2  glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
3                       oclusion_texture, 0);
```

Jetzt müssen die Occluder nur noch entsprechen transformiert gerendert werden.

```
1  auto mvp = glm::mat4{1};
2  auto ligthsize_half = (ligthsize/2.f);
3  mvp = glm::ortho(0.f, float(ligthsize), 0.f, float(ligthsize));
4  mvp *= glm::translate(glm::mat4{1},
5            glm::vec3(-pos.x+ligthsize_half,-pos.y+ligthsize_half,0));
6  render_ocluders(mvp);
```

Dabei wird kein Spezieller Shader benutzt es ist nur wichtig das die Alphakanal Informationen vorhanden sind.

3.2 Shadowmap

Um die 1D Shadowmap zu erzeugen wird ein neuer Framebuffer mit einer Textur erzeugt die nur 1 Pixelhoch ist.

```
1  GLuint shadow1D_texture;
2  glGenTextures(1, &shadow1D_texture);
3
4  glBindTexture(GL_TEXTURE_2D, shadow1D_texture);
5
6  // Give an empty image to OpenGL ( the last "0" )
7  glTexImage2D(GL_TEXTURE_2D, 0,GL_RGBA8, ligthsize,1,
8         0,GL_RGBA, GL_UNSIGNED_BYTE, 0);
9
10 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
11 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
12 //set to repeat so we can oversample the circle
13 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
14 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
15
16
17 glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
18                      shadow1D_texture, 0);
```

Jetzt muss nur noch der Viewport angepasst werden. Der Viewport wird auf die Breite des Lichtes gesetzt so das auf der X-Achse Parallelisiert einen Raycast auf der Occludertextur ausführen kann.

```
1  glBindFramebuffer(GL_FRAMEBUFFER, shadow1D_fbo);
2  glViewport(0,0,ligthsize,1);
3  glClearColor(0.f,0.f,0.f,0.f);
4  glClear(GL_COLOR_BUFFER_BIT);
```

```
5
6  _shadow_mapper_shader->use_shader();
7  GLint id = _shadow_mapper_shader->getUniform("light_resolution");
8  glUniform2f(id,ligthsize,ligthsize);
9  glBindTexture(GL_TEXTURE_2D, oclusion_texture);
10
11
12 glBindVertexArray(quad_VertexArrayID);
13 glDrawArrays(GL_TRIANGLES,0,6);
```

Shader

Der Shader3.1 bekommt die Informationen wie groß die Occlusiontextur ist. Da dieser Shader auf einem Mesh ausgeführt wird der nur einen einen Pixel hoch ist, wird jeder Punkt auf der Occlusiontextur einmal gesampelt und die Tiefen Information in der Textur als Grauwert gespeichert.

Listing 3.1. Shadowmaper Shader

```
1  #version 330
2
3  #define PI 3.14159265359
4  in vec2 var_uv;
5
6  uniform sampler2D ocluder_texture;
7  uniform vec2 light_resolution;
8
9
10 //alpha threshold for our occlusion map
11 const float ALPHA_THRESHOLD = 0.75;
12
13 out vec4 out_color;
14
15 void main(){
16 float distance = 1.0;
17
18 for (float y=0.0; y<light_resolution.y; y+=1.0) {
19 //rectangular to polar filter
20 vec2 norm = vec2(var_uv.s, y/light_resolution.y) * 2.0 - 1.0;
21 float theta = PI*1.5 + norm.x * PI;
22 float r = (1.0 + norm.y) * 0.5;
23
24 //Coordinat which we will sample from occlude map
25 vec2 coord = vec2(-r * sin(theta), -r * cos(theta))/2.0 + 0.5;
26
27 //sample the occlusion map
28 vec4 data = texture2D(ocluder_texture, coord);
29
30 //the current distance is how far from the top we've come
31 float dst = y/light_resolution.y;
32
33 //if we've hit an opaque fragment (occluder), then get new distance
34 //if the new distance is below the current, then we'll use that for our ray
35 float caster = data.a;
36 if(isnan(caster)) caster = 1;
37
38 if (caster > ALPHA_THRESHOLD) {
39 distance = min(distance, dst);
40 }
41 }
42 out_color=vec4(vec3(distance), 1.0);
43 }
```

Abschließend wird die 1D Textur mit noch ein paar weiteren Informationen in einem Vector abgelegt und später grendert zu werden.

```
auto pic = add_image(ligthsize,ligthsize,false);

_ligth_images.emplace_back(std::get<0>(pic),std::get<1>(pic)
                        ,oclusion_texture,shadow1D_texture
                        ,glm::vec3{pos.x-(ligthsize/2.f)
                        ,pos.y-(ligthsize/2.f),0}
                        ,color,glm::vec2(size,size));
```

3.3 Finales Rendering

Letztendlich werden nur noch die Distanz Informationen aus der 1D Textur gelesen und zu Licht mit einem Falloff grendert. Für die bessere Optik wurde noch ein Gausfilter, in der stärke abhängig von der Entfernung zur Quelle, angewendet.

Listing 3.2. Shadow Render Shader

```
#version 330

#define PI 3.14159265359

//inputs from vertex shader
in vec2 var_uv;

//uniform values
uniform sampler2D shadow_map_texture;
uniform vec2 light_resolution;
uniform vec4 Color;

//variable to signal if this light is selected
//and how much blur it should have
uniform float selected;
uniform float blur_factor;


out vec4 out_color;

//sample from the 1D distance map
float sample(float coord, float r) {
return step(r, texture(shadow_map_texture, vec2(coord,0)).r);
}

void main(void) {
//rectangular to polar
vec2 norm = var_uv.st * 2.0 - 1.0;
float theta = atan(norm.y, norm.x);
float r = length(norm);
float coord = 1-(theta + PI) / (2.0*PI);

//the tex coord to sample our 1D lookup texture
//always 0.0 on y axis
vec2 tc = vec2(coord, 0.0);

//the center tex coord, which gives us hard shadows
float center = sample(coord, r);

//we multiply the blur amount by our distance from center
//this leads to more blurriness as the shadow "fades away"
float blur = (1./light_resolution.x) * smoothstep(0., 1., r);

//now we use a simple gaussian blur
float sum = 0.0;

```

```
47  sum += sample(coord - 4.0*blur, r) * 0.05;
48  sum += sample(coord - 3.0*blur, r) * 0.09;
49  sum += sample(coord - 2.0*blur, r) * 0.12;
50  sum += sample(coord - 1.0*blur, r) * 0.15;
51
52  sum += center * 0.16;
53
54  sum += sample(coord + 1.0*blur, r) * 0.15;
55  sum += sample(coord + 2.0*blur, r) * 0.12;
56  sum += sample(coord + 3.0*blur, r) * 0.09;
57  sum += sample(coord + 4.0*blur, r) * 0.05;
58
59  //sum of 1.0 -> in light, 0.0 -> in shadow
60
61  sum = (1-blur_factor)*sum + (blur_factor)*center;
62
63  //multiply the summed amount by our distance, which gives us a radial falloff
64  //then multiply by (light) color
65  out_color = Color * vec4(vec3(1.0)
66                      ,sum * ((1-selected)*smoothstep(1.0, 0.0, r)+(selected)));
67  }
```

4

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

Literaturverzeichnis

CDK02. Coulouris, George, Jean Dollimore und Tim Kindberg: *Verteilte Systeme: Konzepte und Design*. Addison-Wesley-Verlag, 2002.

Che85. Cheriton, David R.: *Preliminary Thoughts on Problem-oriented Shared Memory: A Decentralized Approach to Distributed Systems*, 1985.

Mal97. Malte, Peter: *Replikation in Mobil Computing*. Seminar No 31/1997, Institut für Telematik der Universität Karlsruhe, Karsruhe, 1997. `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=/ira/1997/31`.

Mos93. Mosberger, David: *Memory Consistency Models*. Technical Report 93/11, University of Arizona, November 1993.

A

Glossar

DisASTer	DisASTer (Distributed Algorithms Simulation Terrain), A platform for the Implementation of Distributed Algorithms
DSM	Distributed Shared Memory
AC	Linearisierbarkeit (atomic consistency)
SC	Sequentielle Konsistenz (sequential consistency)
WC	Schwache Konsistenz (weak consistency)
RC	Freigabekonsistenz (release consistency)

B

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

Datum	Unterschrift der Kandidatin / des Kandidaten