



HOCHSCHULE TRIER

Trier University of Applied Sciences

Informatik - Computer Science

Titel der Arbeit auf Deutsch

Jeremias Boos

Hausarbeit zur Vorlesung Realtime Rendering

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, 11. März 2016

Kurzfassung

Dies ist eine Dokumentation zur Implementierung eines Verfahrens zur Berechnung von 2D Schatten. Das hier vorgestellte Verfahren arbeitet hauptsächlich auf der GPU mit Hilfe von Framebuffern und Fragment Shadern.

Inhaltsverzeichnis

1	Motivation	1
2	Verfahren	2
2.1	Oclusionmap	2
2.2	Sample Distanz	2
2.3	Render Licht	3
3	Implementierung	6
3.1	Oclusionmap	6
3.2	Shadowmap	7
3.3	Finales Rendering	9
4	Zusammenfassung und Ausblick	11
	Literaturverzeichnis	12

Motivation

Mich hat es schon immer interessiert wie Schatten in 2D Spielen umgesetzt werden könnten. Zu diesem Thema habe ich zwei verfahren gefunden. Eines der Verfahren basiert auf Raycasts die mit Hilfe einer Physikengine einen “Lichtmesh“erzeugt und stark CPU-lastig ist. 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 das GPU basierte Verfahren entschieden.

Verfahren

Das Verfahren zur Schattenberechnung besteht grundsätzlich aus 3 Schritten. Die verwendete Methodik ist ein Raycastverfahren das durch einige mathematische Umformungen der Texturen den Raycast stark vereinfacht.

2.1 Occlusionmap

Zur Licht Berechnung muss zuerst eine Occlusionmap erstellt werden. Diese Textur bildet ab welchen Bereich das Licht vom Level abdeckt.



Abb. 2.1. Testgrafik

2.2 Sample Distanz

Die Occlusionmap wird zur besseren Parallelisierbarkeit in Polarkoordinaten gesampelt.

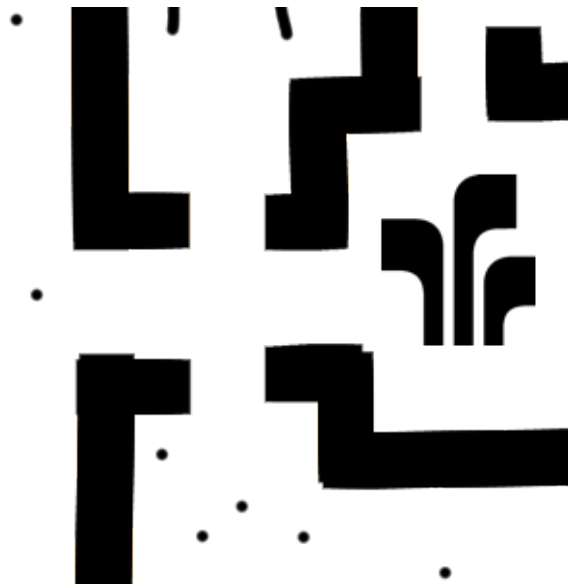


Abb. 2.2. Occlusionmap



Abb. 2.3. Occlusionmap in polar Koordinaten

So können wir einen Raycast für 360 um die Lichtquelle auf der X Achse parallelisieren. Hier wird nun für jedes X eine Distanz bis zum ersten blockierenden Pixel gemessen.^{2.4} Die gemessene Distanz wird in einer 1D Textur zur weiter Verarbeitung gespeichert.^{2.5}

2.3 Render Licht

Im letzten Schritt wird ein Sprite in der Größe des Lichtes gerendert. Durch Verwendung einer Stepfunktion wird die Distanz zum mittl Punkt mit der maximal

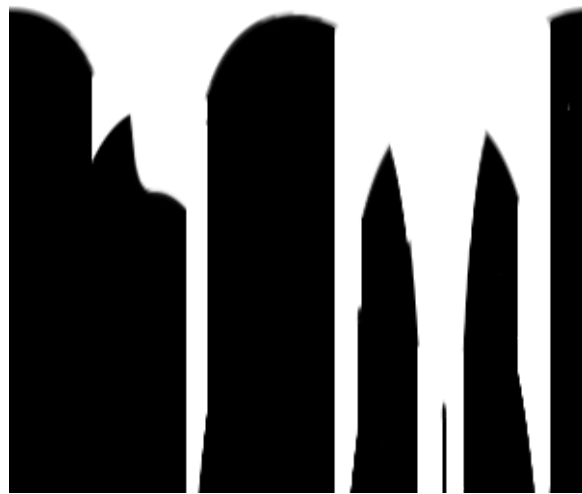


Abb. 2.4. Symbolische Shadowmap in polar Koordinaten



Abb. 2.5. Gesampelte Distanzdaten.

Entfernung die in der Textur gespeichert wurde verglichen. Wenn der gesampelte Punkt weiter weg ist als die Gemessene Entfernung liegt der Pixel im Schatten und wird schwarz gezeichnet. Anschließend muss das Sprite nur noch additiv Geblendet gerendert werden.^{2.7}



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. Ohne Verwendung von Framebuffers können die entsprechenden Texturen nicht erstellt werden die zu Berechnung der Schatten benötigt werden.

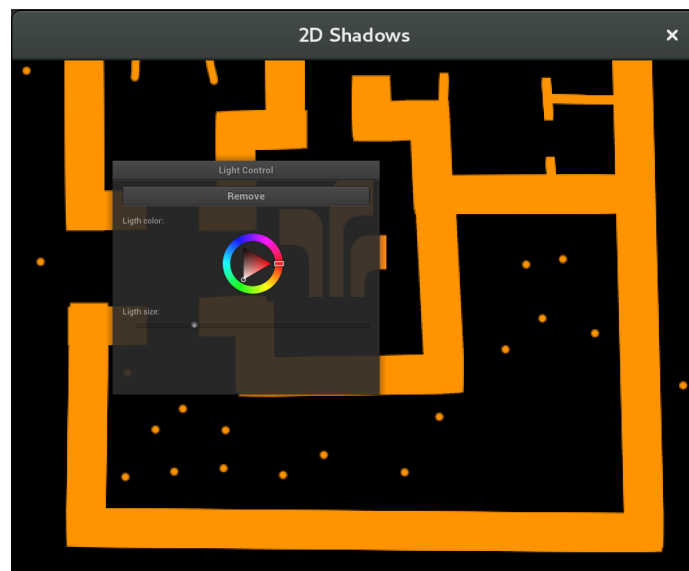


Abb. 3.1. Programm beim nach dem Start.

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

3.1 Oclusionmap

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

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

```

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

```

Diese wird nun an ein FBO gebunden.

```

1 // Set "occlusion_texture" as our colour attachment #0
2 glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
3                     occlusion_texture, 0);

```

Jetzt werden die Occluder mit der entsprechen Transformierung gerendert um den vom Licht abgedeckten Bereich zu erfassen.

```

1 auto mvp = glm::mat4{1};
2 auto lighsize_half = (lighsize/2.f);
3 mvp = glm::ortho(0.f, float(lighsize), 0.f, float(lighsize));
4 mvp *= glm::translate(glm::mat4{1},
5                       glm::vec3(-pos.x+lighsize_half,-pos.y+lighsize_half,0));
6 render_occluders(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, lighsize,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,lighsize,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, lightsize, lightsize);
9  glBindTexture(GL_TEXTURE_2D, occlusion_texture);
10
11
12  glBindVertexArray(quad_VertexArrayID);
13  glDrawArrays(GL_TRIANGLES, 0, 6);

```

Shader

Der Shader3.1 bekommt die Informationen wie groß die Occlusionstextur ist. Da dieser Shader auf einem Mesh ausgeführt wird der nur einen Pixel hoch ist, wird jeder Punkt auf der Occlusionstextur exakt nur einmal gesampelt und die Tiefen Information in der 1D 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 um später grendert zu werden.

```

1 auto pic = add_image(ligthsized, ligthsized, false);
2
3 _ligth_images.emplace_back(std::get<0>(pic), std::get<1>(pic)
4                             , occlusion_texture, shadow1D_texture
5                             , glm::vec3{pos.x - (ligthsized / 2.f)
6                             , pos.y - (ligthsized / 2.f), 0}
7                             , color, glm::vec2(size, size));

```

3.3 Finales Rendering

Beim Rendern der Schatten wird letztendlich nur noch die Distanz Informationen aus der 1D Textur gelesen und als Licht mit einem Falloff gerendert. Für die bessere Optik wurde noch ein Gausfilter, in der stärke abhängig von der Entfernung zur Quelle, angewendet.

Listing 3.2. Shadow Render Shader

```

1 #version 330
2
3 #define PI 3.14159265359
4
5 //inputs from vertex shader
6 in vec2 var_uv;
7
8 //uniform values
9 uniform sampler2D shadow_map_texture;
10 uniform vec2 light_resolution;
11 uniform vec4 Color;
12
13 //variable to signal if this light is selected
14 //and how much blur it should have
15 uniform float selected;
16 uniform float blur_factor;
17
18
19 out vec4 out_color;
20
21 //sample from the 1D distance map
22 float sample(float coord, float r) {
23     return step(r, texture(shadow_map_texture, vec2(coord, 0)).r);
24 }
25
26 void main(void) {
27     //rectangular to polar
28     vec2 norm = var_uv.st * 2.0 - 1.0;
29     float theta = atan(norm.y, norm.x);
30     float r = length(norm);
31     float coord = 1 - (theta + PI) / (2.0 * PI);
32
33     //the tex coord to sample our 1D lookup texture
34     //always 0.0 on y axis
35     vec2 tc = vec2(coord, 0.0);
36
37     //the center tex coord, which gives us hard shadows
38     float center = sample(coord, r);
39
40     //we multiply the blur amount by our distance from center
41     //this leads to more blurriness as the shadow "fades away"
42     float blur = (1. / light_resolution.x) * smoothstep(0., 1., r);
43
44     //now we use a simple gaussian blur
45     float sum = 0.0;

```

```
46
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 }
```

Zusammenfassung und Ausblick

In dem Projekt wurde erfolgreich das Verfahren zur 2D Schattenberechnung umgesetzt. Bei der Umsetzung war vor allem das arbeiten mit Framebuffern eine neue Erfahrung. Die aktuelle Implementierung sehr rudimentäre und enthält nur die reine Schatten Berechnung. Gute Erweiterungen wären zum Beispiel Unterstützung für Normalen und eine Schattierung der Occluder. Verbesserungsmöglichkeiten Beständen auch zusätzlich im Bereich des Rendering, in der aktuellen Version wird für jedes Licht ein eigenes Mesh erzeugt und separat gerendert. Was zu einer extrem hohen Pixelfüllrate bei vielen Lichtern führt. Die Speicherung der 1D Texturen könnte auch noch durch Atlase verbessert werden.

In der Zukunft würde ich die umgesetzte Technik gerne noch weiter ausbauen und verbessern. Ziel ist es die Technik soweit zu entwickeln das sie in 2D Spielen eingesetzt werden kann. // // Abschließend noch vielen dank an Matt DesLauriers. Diese Implementierung basiert stark auf seiner Arbeit auf github.[Des]

Literaturverzeichnis

Des. DESLAURIERS, MATT: *2D Pixel Perfect Shadows*.