Hochschule Harz

Analysis of the lighting in old video games using the example of the "Dark Engine"

Alexander Johr u34584 m27007

Examiner: Prof. Daniel Ackermann

Wernigerode, D-38855

28th of February 2019

Harz University of Applied Sciences

Analysis of the lighting in old video games using the example of the "Dark Engine"

Old game engines have accomplished astonishing things in the past and created loving communities that cherish those games to this day.

Modern game engines have improved a lot compared to those older engines, yet those communities still stick to the older technology despite having their limitations.

One great example of this is the so-called *Dark Engine* used by games of the *Thief* series. Although this series is 20 years old at the time of this writing it still looks very appealing and the communities surrounding those games are creating fan missions for it to this day.

This analysis is focused on how the *Dark Engine* achieved its visual fidelity through techniques like light baking and how it is different from modern game engines. It also tries to determine what the limitations of this engine were and why they were needed.

Prof. Daniel Ackermann
Examiner

Contents

List of Figures

List of Listings

1 Introduction

1.1 Dark Engine

The so-called *Dark Engine* is a game engine developed by *Looking Glass Studios* for the game *Thief: The Dark Project* and *Thief II: The Metal Age*. It was also reused by *Irrational Games* and *Looking Glass Studios* for the game *System Shock 2*.[1] The renderer was developed by Sean Barrett with inspiration from the *Quake* engine.[2] Levels for the *Dark Engine* can be created with the *Dromed* editor.

1.1.1 Dromed

Dromed is the level editor for the *Dark Engine*. It allows creating levels for *Thief: The Dark Project*, *Thief Gold* and *Thief II: The Metal Age* as well as *System Shock 2*. The origin of the name *Dromed* has its roots in the former working title of the game *Thief: The Dark Project*. The game was initially planned to be a sword-combat action game based on the Arthurian legend called *Dark Camelot*.[3] Dromed is a play on words: Camelot is shortened to camel, which was associated with and changed to dromedary and is shortened yet again to *Dromed*.

Even though the engine is from 1998[4], it is still very popular in the *Thief* community and various contests are being held for new content creation for the game. Getting started with the *Dromed* editor has a steep learning curve compared to modern game engines, but creators are willing to pay that price. At the time of writing, the latest contest was the *Thief II: The Metal Age 20th Anniversary Contest* celebrating the North American release of *Thief II: The Metal Age* on March 23rd, 2000.[5]

1.1.2 Brushes

The environment of levels built with *Dromed* are created by using so-called brushes (See figure A.1. Brushes are highlighted in grey and cyan lines: one cube and two wedges).[6]

Brushes are simple polyhedra like cubes, cylinders, pyramids, wedges and dodecahedron. Brushes fill the world with solid matter, water or air and carve out of existing brushes.[7]

[1] cf. Grossman, Postmortems from game developer, p. 7
[2] cf. Sean Barrett, The 3D Software Rendering Technology of 1998's Thief: The Dark Project
[3] cf. Grossman, Postmortems from game developer, p. 171
[4] cf. Grossman, Postmortems from game developer, p. 172
[5] cf. The TTLG Forums Community, Thief II: The Metal Age 20th Anniversary Contest
[6] cf. Sean Barrett, The 3D Software Rendering Technology of 1998's Thief: The Dark Project
[7] cf. Sean Barrett, The 3D Software Rendering Technology of 1998's Thief: The Dark Project

Those brushes are then compiled into a cell and portal system[8] which is described in the dissertation of Seth Jared Teller "Visibility Computations in Densely Occluded Polyhedral Environments".[9]

The entire level is subdivided by occluders like walls, ceilings and floors into cells.[10] Every transparent boundary of any cell with any other adjacent cell is called a portal.[11] For each cell it is computed which other cells are potentially visible through those portals. This is called the potentially visible set (PVS).[12]

2 Methods

2.1 Finding the correct profiler to analyze the *Dark Engine*

When analyzing *Dark Engine* games they have to be considered as a black box. There are rumors that the source code was found, but the complete source code is not available online. The website *thepiratebay* lists one torrent for the *Dark Engine* source code. During the entire time of writing this document, the author tried to download the source code from this torrent, but it never passed 98.1%.

This means, to analyze how the lightbaking works in the *Dark Engine*, profiling has to be done by software, which can for example intercept the calls made from the CPU to the GPU.

2.1.1 NVIDIA® Nsight™

The first tool, that was tested to profile the game was *NVIDIA® Nsight™*. It allows to start a game from its own launcher and then intercepts the API calls sent by the CPU to the GPU through the graphics API. The supported APIs are *DirectX 11* and *12*, *OpenGL* and *Vulkan*. The *Dark Engine* was originally developed for *DirectX 6* and was upgraded to *DirectX 9*, which means it can't be profiled with *Nsight™*. However, there is a wrapper called *dgVoodoo* which makes it possible for old graphics APIs to be interoperable with modern APIs. Even after wrapping *DirectX 9* with *DirectX 11* using *dgVoodoo* it was not possible to profile the game with *Nsight™*. For that reason, another tool for profiling older graphics APIs was researched.

[8] cf. Sean Barrett, Interview with Sean Barrett
[9] cf. Sean Barrett, The 3D Software Rendering Technology of 1998's Thief: The Dark Project
[10] cf. Teller, "Visibility Computations in Densely Occluded Polyhedral Environments," p.3
[11] cf. Teller, "Visibility Computations in Densely Occluded Polyhedral Environments," p.3
[12] cf. Teller, "Visibility Computations in Densely Occluded Polyhedral Environments," p.155

2.1.2 PIX

It turned out that there is a tool called *PIX* which is part of the *DirectX SDK*. Launching *Dark Engine* games with the *PIX* launcher worked immediately and allowed to capture frames. *PIX* then provided the selection of each frame and made it possible to search for all loaded textures in the *Objects* window and all API calls made in the *Events* window (see figure B.1).

Therefore *PIX* from the *DirectX 9 SDK* was used for this analysis. Another version for newer APIs like *DirectX 12* exists, but those only capture 64-bit processes. Games made with the *Dark Engine*, however, are 32-bit processes.

2.2 Creating a test scene

In order to analyze how lightbaking and dynamic lighting works in the *Dark Engine* a scene was created. For that the *Dromed* editor of the installation of *Thief II: The Metal Age* was used. The scene consists of a room with a slanted ceiling. A bed was positioned as well to determine how objects influence the lightmaps and how the objects themselves are being lit by light sources. The room contains three torches, each one's light radius overlapping the radius of both neighboring torches and each one configured with a different prime color: red, green and blue as depicted in figure E.2.

2.3 Analysis of well-defined recorded frames

The test scene was launched with *PIX* and for each combination of lit and unlit torches a frame was recorded.

Each frame was then analyzed for the following information:

- How are the lightmaps stored?
- How are the lightmaps applied to the meshes?
- What happens when a light source changes its state?
- And how are objects lit?

3 Results

3.1 Lightmaps

Lightmaps are only created for surfaces generated by brushes. Although lightmaps are not created for objects, they at least manipulate the baked lightmaps by casting shadows. For example, the shadow of the bed in the scene as depicted in figure E.2.

The first frame was analyzed for the draw calls made to render the walls with the light from the torches (see figure C.1 and listing C.2).

The calls *SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG1)* and *SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_DISABLE)* are called to merely use the raw RGB-value of the texture and disable alpha operations respectively[13,14,15]. Both calls refer to the texture with the index *0*.

Next, the call *SetTexture(0, 0x0C16E110)* sets the texture with the index *0*. Clicking on the reference *0x0C16E110* opens the loaded texture in the *Details* window (see C.3) which contains the texture of the wallpaper.

Following this, the second texture is configured with *SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_MODULATE2X)* to multiply the RGB-values of the second texture with the values of the first texture and doubling the value by shifting the result one bit to the left to brighten the result[16,17,18].

Now the second texture is loaded by calling *SetTexture(1, 0x0C13EAA0)*. Again, clicking the reference *0x0C13EAA0* opens the texture in the *Details* window (see figure C.5). This time it is the baked lightmap.

Lastly, the call *DrawPrimitiveUP(D3DPT_TRIANGLELIST, 9, 0x0ED04C74, 40)* draws the mesh with the two textures. Both textures are now blended together as depicted in the render output C.4. The *Mesh* tab of the *Details* window shows, that there is no vertex lighting because all vertices are colored white: D3DCOLOR_ARGB(0xff,0xff,0xff,0xff) (see figure C.6).

The same analysis was repeated for each of the different frames with the different combinations of lit and unlit torches.

[13] cf. Direct3D 9 Graphics - SetTextureStageState method
[14] cf. Direct3D 9 Graphics - D3DTEXTUREOP enumeration
[15] cf. Direct3D 9 Graphics - D3DTEXTURESTAGESTATETYPE enumeration
[16] cf. Direct3D 9 Graphics - SetTextureStageState method
[17] cf. Direct3D 9 Graphics - D3DTEXTUREOP enumeration
[18] cf. Direct3D 9 Graphics - D3DTEXTURESTAGESTATETYPE enumeration

It turned out that as the torches were extinguished or lit, the lightmaps were swapped, although they kept the same reference. It looks like, that whenever a light source changes its state, the correct lightmap is looked up in a table of all possible combinations of those light source states.

Appendix D shows all of those lightmaps with their corresponding rendered frame. The texel color of the overlapping areas are correctly blended together for all combinations. That indicates that each combination of active and inactive light sources generates a unique set of lightmaps for the environment.

This also means that the number of lightmaps increases exponentially with each light source. This is also reflected by the increasing filesize of the *.mis*-files which contain the level data.

3.2 Realtime lighting

Objects also react to extinguishing or lighting torches. The bed positioned to the bottom right of the red torch is drawn in consecutive draw calls for each of the different textures. The first frame was analyzed for the draw calls which rendered the bed headboard (see listing D.1).

Before a mesh from an object is drawn the texture is set again, but this time the lightmap texture is disabled by setting it to *null* with the call *SetTexture(1, NULL)*.

Again, *SetTexture(0, 0x0C16E110)* sets the texture with the index *0*. The *Details* window shows, that the texture of the bed headboard is loaded (see D.2).

The next *DrawPrimitiveUP* draws the bed headboard as depicted in figure D.3. The *Details* window revealed how the lighting is done.

The *Mesh* tab shows the information of the vertices position, diffuse and specular color and UV coordinates. Figure D.4 shows, that the second argument - the red value - varies with each record of the diffuse color, while all other values remain the same. Furthermore, this time there is only one texture coordinate, indicating that the lighting is entirely done with vertex lighting and no lightmap is used for these objects.

Two properties seem to influence the color value:

- The closer the vertex is to the torch
- and the more its normal is facing it.

4 Discussion

Game engines were largely optimized for ease of use. They don't share the limitations of older engines, but perform pretty well nonetheless. But those same limitations enabled game engines from the past to accomplish astonishing things where modern game engines took a step back.

Level designers don't need to think about occlusion culling when creating their environments. They can create their levels in third-party software like *Maya*, *3ds Max* or *Blender* and import them later on to *Unity* or *Unreal Engine*. Information about occlusion culling can be generated by those engines without any involvement of the level designer. It simply has to be enabled.[19,20] If they want to improve performance of occlusion culling any further, the game engines enable them to do so with components like Occlusion Areas in Unity.[21]

Light baking in modern game engines tends to consume a lot of time even with the vast advancements in processing power. This is where game engines like the *Dark Engine* can really shine. The *Dark Engine* only supports baking lightmaps for meshes generated by brushes and it forces the level designer to use those. But at the same time, light baking is blazing fast and it is generated for every combination of active and inactive light sources. That is something that modern game engines can't compete with and it might be one of the reasons why the community didn't move on but are creating more content in this old game engine.

Modern hardware and game engines can accomplish similar effects with realtime lighting, but there is no question that lighting done with baked lightmaps looks more appealing. Scripting in *Unity* enables the programmer to implement their own switching of lightmaps at runtime[22] but the *Dark Engine* makes it so convenient users do not have to worry about any of that.

In conclusion, it might be worth implementing a mechanism into modern game engines that allows to automatically generate multiple lightmaps for each desired lighting condition. It is also possible that advancements in graphic hardware and technologies like realtime raytracing will advance to a state that makes baked lightmaps obsolete.

[19] cf. Unity Technologies, Unity Manual - Occlusion culling
[20] cf. Epic Games, Unreal Engine 4 Documentation - Visibility and Occlusion Culling
[21] cf. Unity Technologies, Unity Manual - Occlusion Areas
[22] cf. Unity Technologies, Unity Manual - LightmapSettings

References

Direct3D 9 Graphics - D3DTEXTUREOP enumeration. Url: https://docs.microsoft.com/en-us/windows/win32/direct3d9/d3dtextureop, last access February 27th 2020.

Direct3D 9 Graphics - D3DTEXTURESTAGESTATETYPE enumeration. Url: https://docs.microsoft.com/de-de/windows/win32/direct3d9/d3dtexturestagestatetype, last access February 27th 2020.

Direct3D 9 Graphics - SetTextureStageState method. Url: https://docs.microsoft.com/en-us/windows/win32/api/d3d9helper/nf-d3d9helper-idirect3ddevice9-settexturestagestate, last access February 27th 2020.

Epic Games.

Unreal Engine 4 Documentation - Visibility and Occlusion Culling. Url: https://web.archive.org/web/20190704003501/https://docs.unrealengine.com/en-US/Engine/Rendering/VisibilityCulling/index.html, last access February 28th 2020.

Grossman, Austin.

Postmortems from game developer. Taylor & Francis, 2003.

Sean Barrett.

Interview with Sean Barrett. Url: https://web.archive.org/web/20090511181005/http://southquarter.com:80/2009/05/08/interview-with-sean-barrett, last access February 28th 2020.

— The 3D Software Rendering Technology of 1998's Thief: The Dark Project. Url: https://web.archive.org/web/20200214115759/https://nothings.org/gamedev/thief_rendering.html, last access February 28th 2020.

Teller, Seth Jared.

"Visibility Computations in Densely Occluded Polyhedral Environments." PhD thesis. EECS Department, University of California, Berkeley, Oct. 1992. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/6250.html.

The TTLG Forums Community.

Thief II: The Metal Age 20th Anniversary Contest. Url: https://web.archive.org/web/20200227001352/https://www.ttlg.com/forums/showthread.php?t=149648, last access February 27th 2020.

Unity Technologies.

Unity Manual - LightmapSettings. Url: https://web.archive.org/web/20200228225708/https://docs.unity3d.com/2019.2/Documentation/ScriptReference/LightmapSettings.html, last access February 28th 2020.

— Unity Manual - Occlusion Areas. Url: https://web.archive.org/save/https://docs.unity3d.com/2019.2/Documentation/Manual/class-OcclusionArea.html, last access February 28th 2020.

Unity Technologies.

Unity Manual - Occlusion culling. Url: https://web.archive.org/web/20190909002950/https://docs.unity3d.com/Manual/OcclusionCulling.html, last access February 28th 2020.

A Dromed

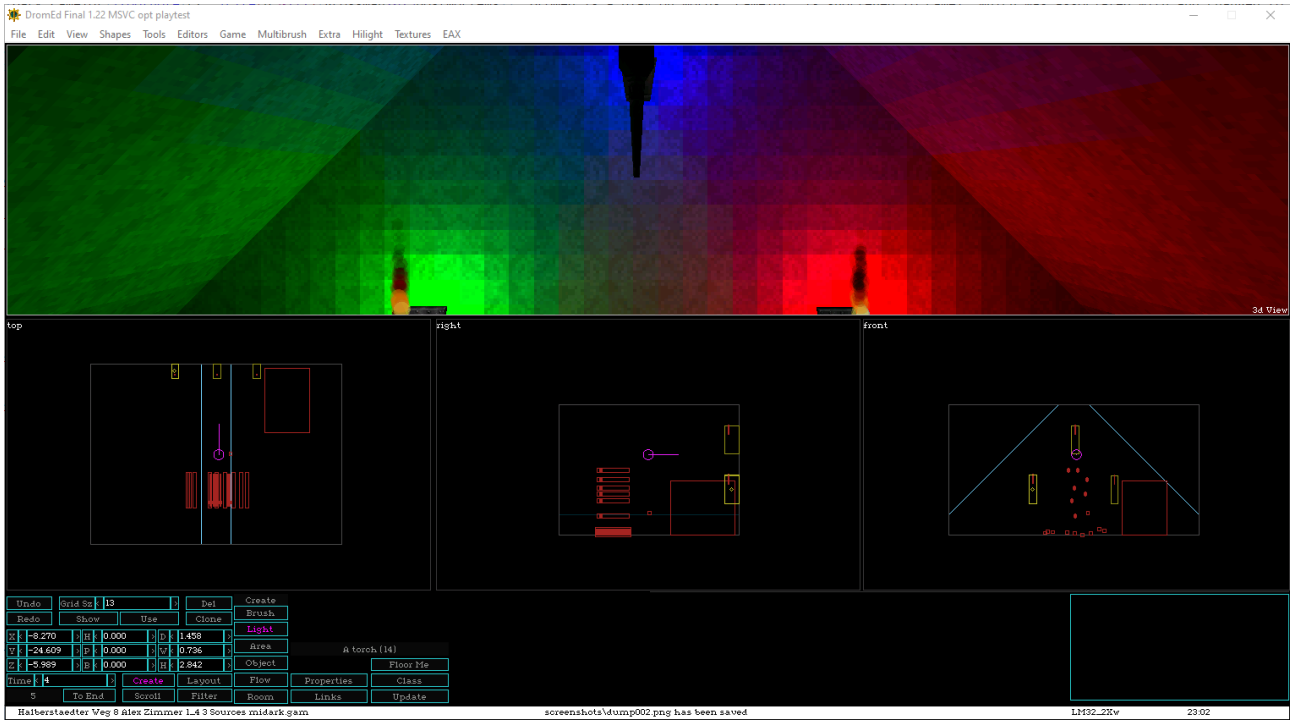


Figure A.1: Overview of the Dromed editor

B PIX overview

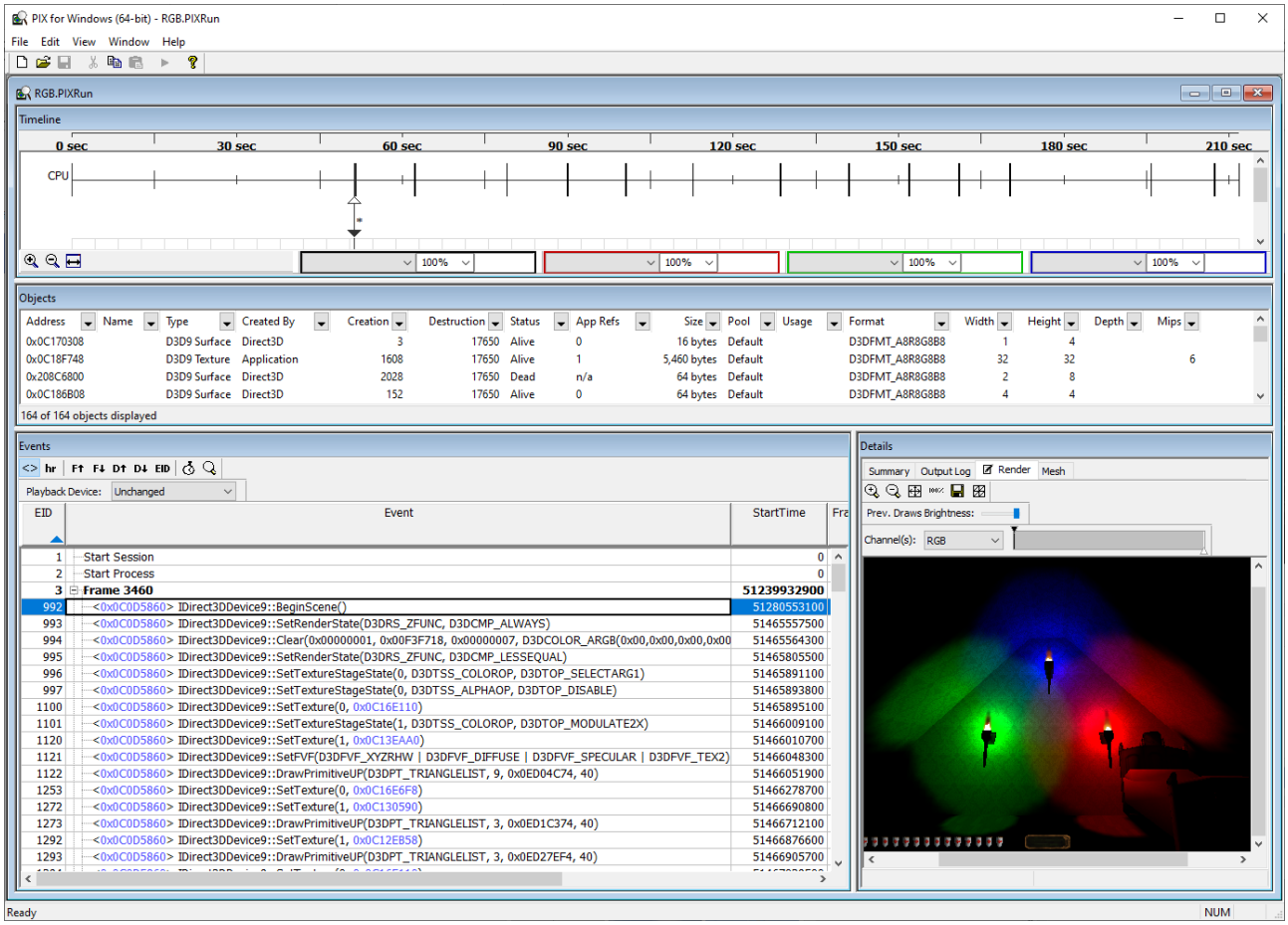


Figure B.1: Overview of the PIX user interface

C Draw calls with lightmaps

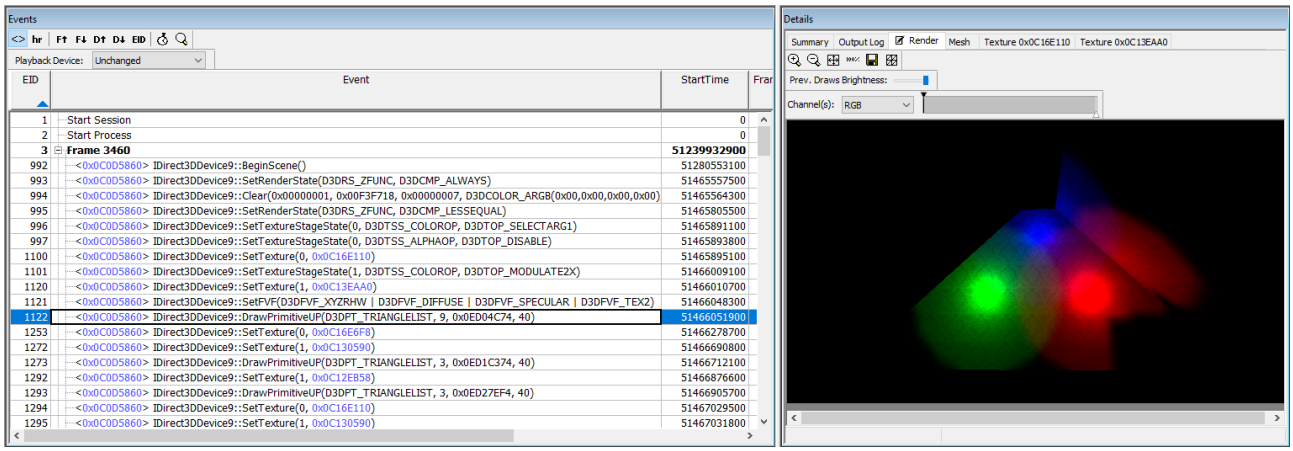


Figure C.1: First draw call of the first frame

```
BeginScene()
SetRenderState(D3DRS_ZFUNC, D3DCMP_ALWAYS)
Clear(0x00000001, 0x00F3F718, 0x00000007
      , D3DCOLOR_ARGB(0x00,0x00,0x00,0x00), 1.000f, 0x00000000)
SetRenderState(D3DRS_ZFUNC, D3DCMP_LESSEQUAL)
SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG1)
SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_DISABLE)
SetTexture(0, 0x0C16E110)
SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_MODULATE2X)
SetTexture(1, 0x0C13EAA0)
SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_SPECULAR | D3DFVF_TEX2)
DrawPrimitiveUP(D3DPT_TRIANGLELIST, 9, 0x0ED04C74, 40)
SetTexture(0, 0x0C16E6F8)
SetTexture(1, 0x0C130590)
DrawPrimitiveUP(D3DPT_TRIANGLELIST, 3, 0x0ED1C374, 40)
```

Listing C.2: First draw call of the first frame

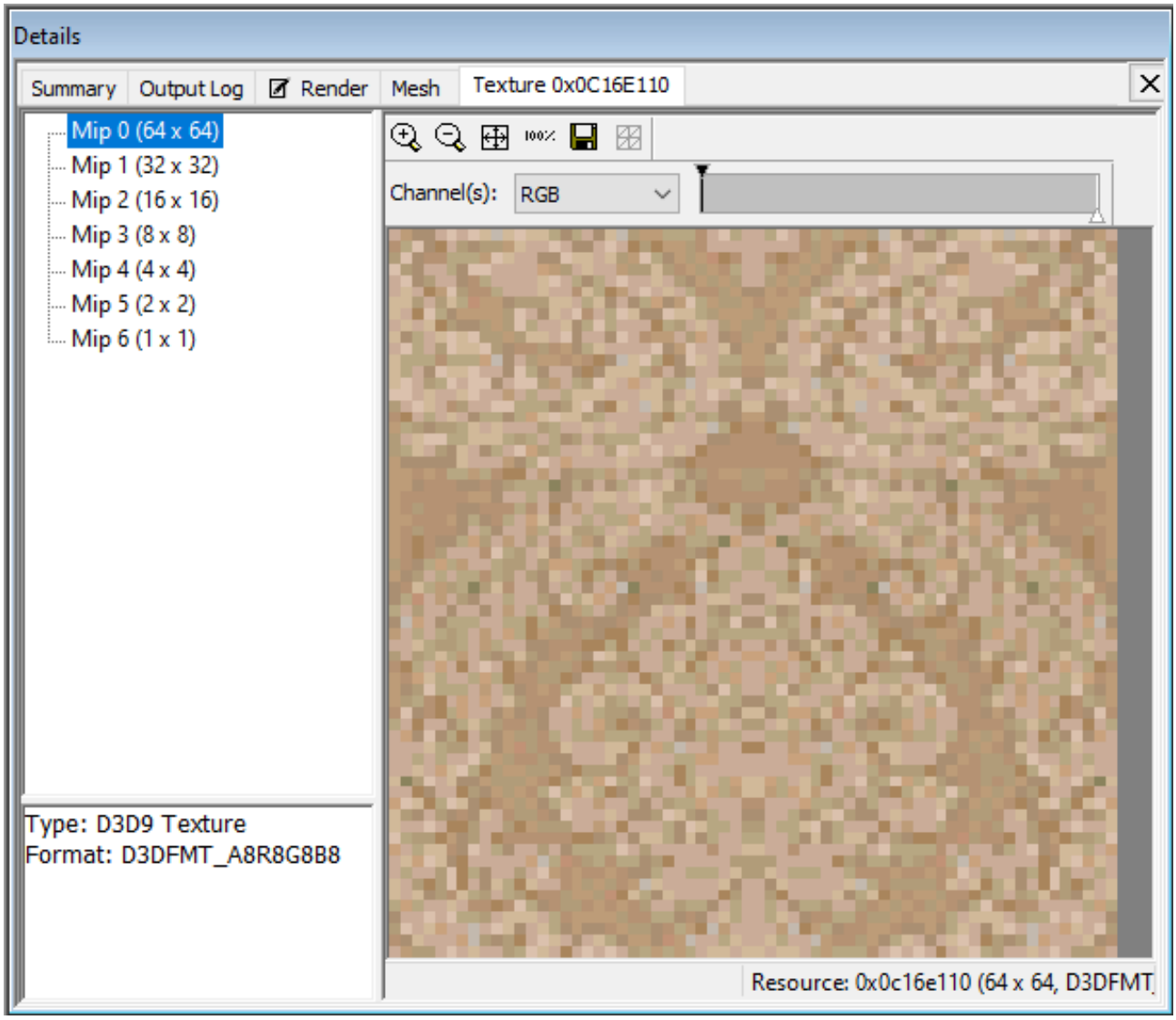


Figure C.3: Details of the wallpaper texture

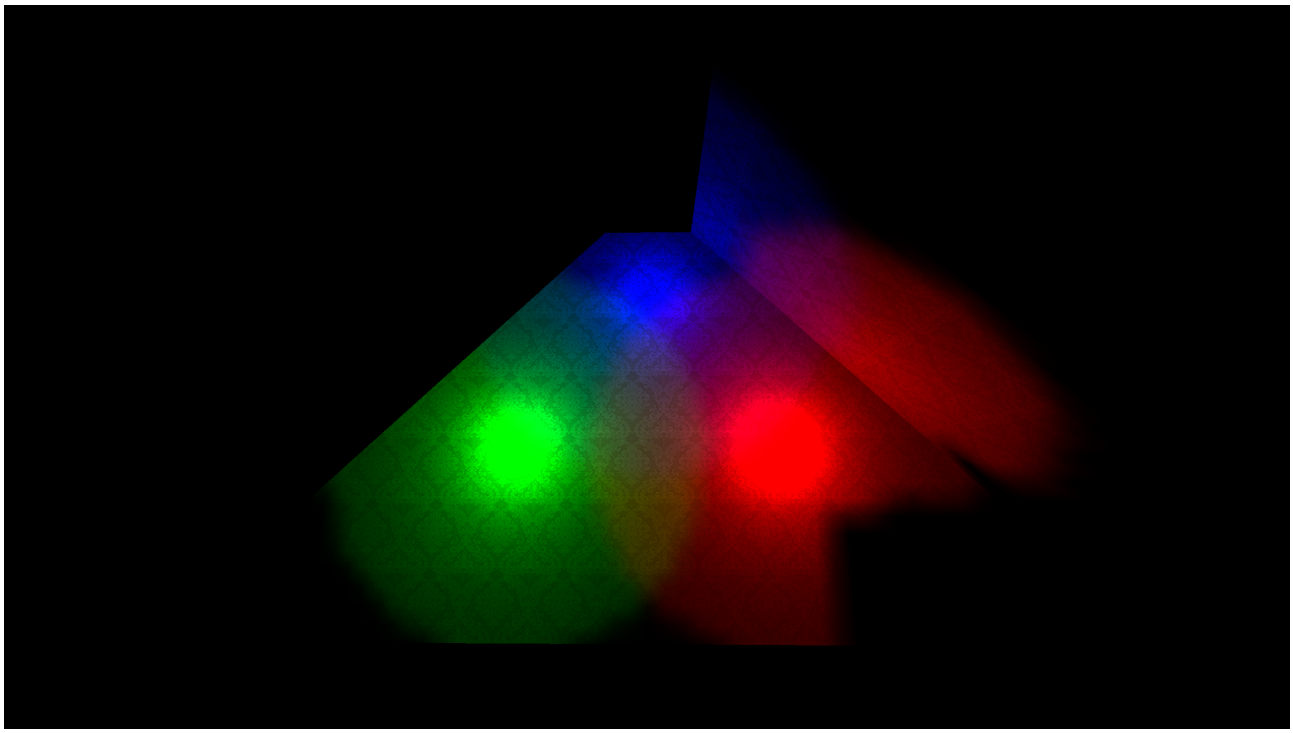


Figure C.4: Render of mesh with lightmap texture

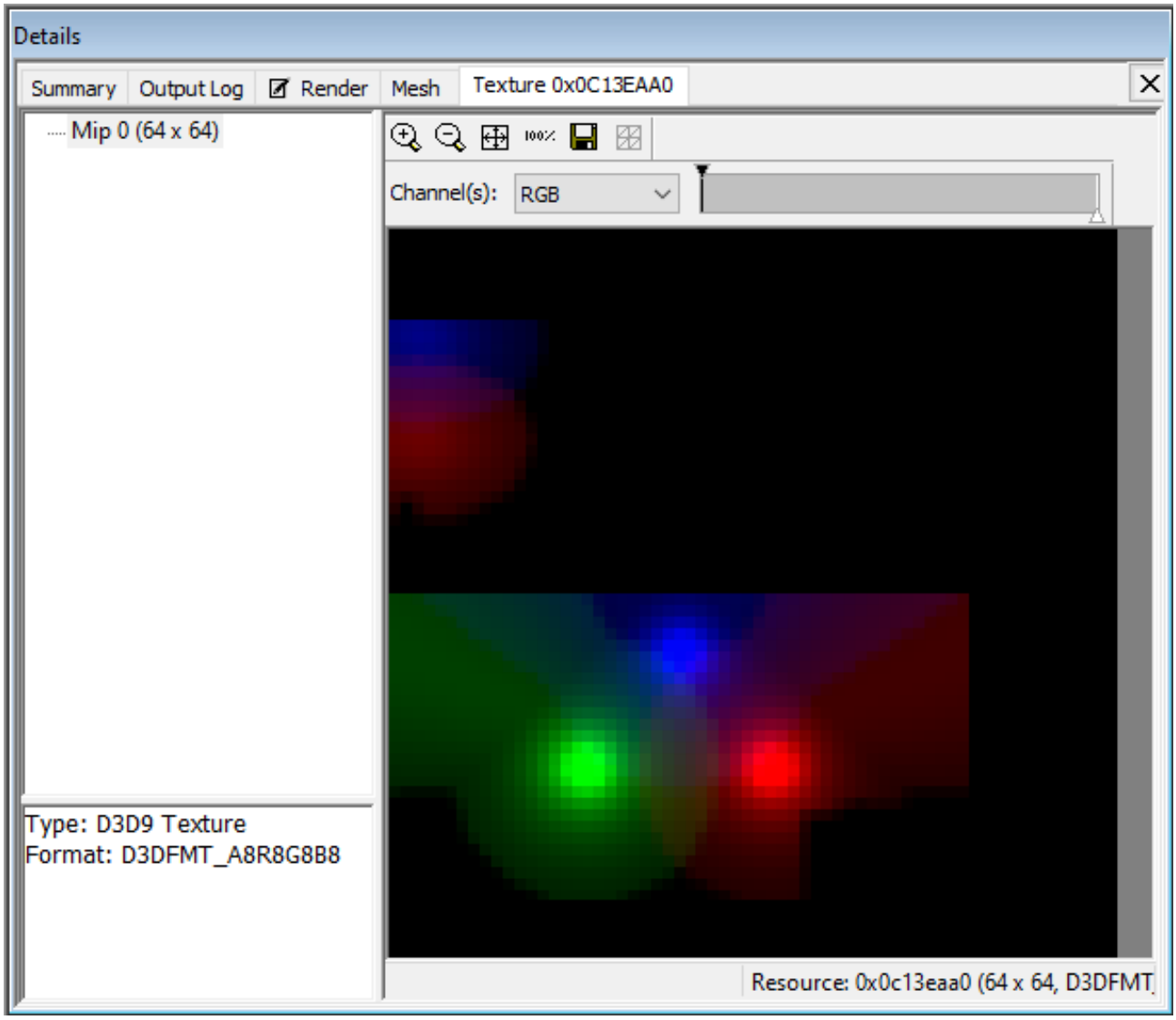
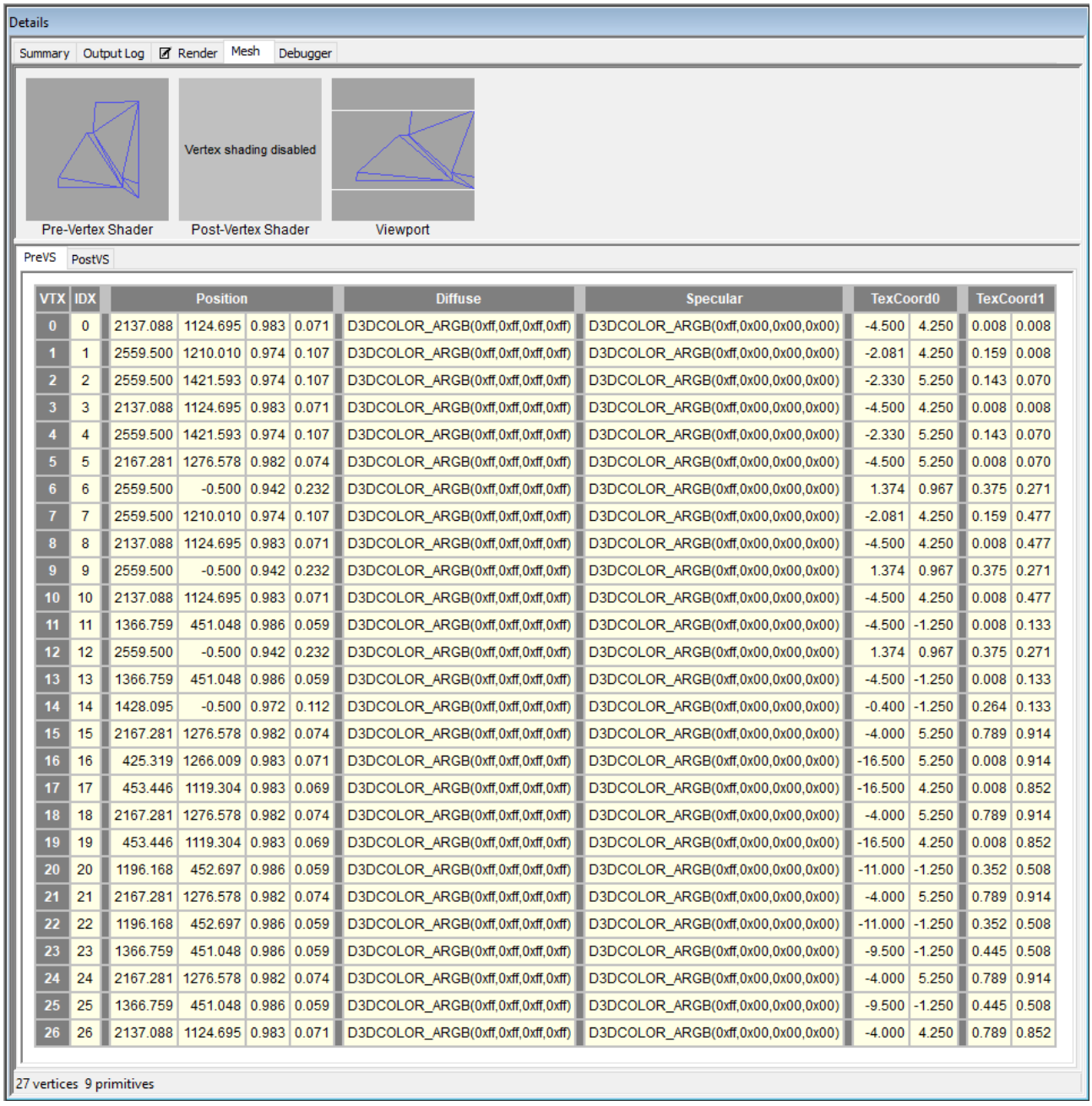


Figure C.5: Details of the lightmap texture



VTX	IDX	Position				Diffuse	Specular	TexCoord0		TexCoord1	
0	0	2137.088	1124.695	0.983	0.071	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	4.250	0.008	0.008
1	1	2559.500	1210.010	0.974	0.107	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-2.081	4.250	0.159	0.008
2	2	2559.500	1421.593	0.974	0.107	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-2.330	5.250	0.143	0.070
3	3	2137.088	1124.695	0.983	0.071	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	4.250	0.008	0.008
4	4	2559.500	1421.593	0.974	0.107	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-2.330	5.250	0.143	0.070
5	5	2167.281	1276.578	0.982	0.074	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	5.250	0.008	0.070
6	6	2559.500	-0.500	0.942	0.232	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	1.374	0.967	0.375	0.271
7	7	2559.500	1210.010	0.974	0.107	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-2.081	4.250	0.159	0.477
8	8	2137.088	1124.695	0.983	0.071	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	4.250	0.008	0.477
9	9	2559.500	-0.500	0.942	0.232	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	1.374	0.967	0.375	0.271
10	10	2137.088	1124.695	0.983	0.071	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	4.250	0.008	0.477
11	11	1366.759	451.048	0.986	0.059	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	-1.250	0.008	0.133
12	12	2559.500	-0.500	0.942	0.232	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	1.374	0.967	0.375	0.271
13	13	1366.759	451.048	0.986	0.059	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.500	-1.250	0.008	0.133
14	14	1428.095	-0.500	0.972	0.112	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-0.400	-1.250	0.264	0.133
15	15	2167.281	1276.578	0.982	0.074	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.000	5.250	0.789	0.914
16	16	425.319	1266.009	0.983	0.071	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-16.500	5.250	0.008	0.914
17	17	453.446	1119.304	0.983	0.069	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-16.500	4.250	0.008	0.852
18	18	2167.281	1276.578	0.982	0.074	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.000	5.250	0.789	0.914
19	19	453.446	1119.304	0.983	0.069	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-16.500	4.250	0.008	0.852
20	20	1196.168	452.697	0.986	0.059	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-11.000	-1.250	0.352	0.508
21	21	2167.281	1276.578	0.982	0.074	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.000	5.250	0.789	0.914
22	22	1196.168	452.697	0.986	0.059	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-11.000	-1.250	0.352	0.508
23	23	1366.759	451.048	0.986	0.059	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-9.500	-1.250	0.445	0.508
24	24	2167.281	1276.578	0.982	0.074	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.000	5.250	0.789	0.914
25	25	1366.759	451.048	0.986	0.059	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-9.500	-1.250	0.445	0.508
26	26	2137.088	1124.695	0.983	0.071	D3DCOLOR_ARGB(0xff,0xff,0xff,0xff)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	-4.000	4.250	0.789	0.852

Figure C.6: Details of mesh with lightmap texture

D Draw calls with vertex lighting

```
SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE)
SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_MODULATE)
SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_DISABLE)
SetTexture(0, 0x0C16E3B0)
SetTexture(1, NULL)
SetTexture(0, 0x0C16E8F0)
SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_SPECULAR | D3DFVF_TEX1)
DrawPrimitiveUP(D3DPT_TRIANGLELIST, 8, 0x0ED1C374, 32)
```

Listing D.1: Draw calls for the first vertex lit mesh

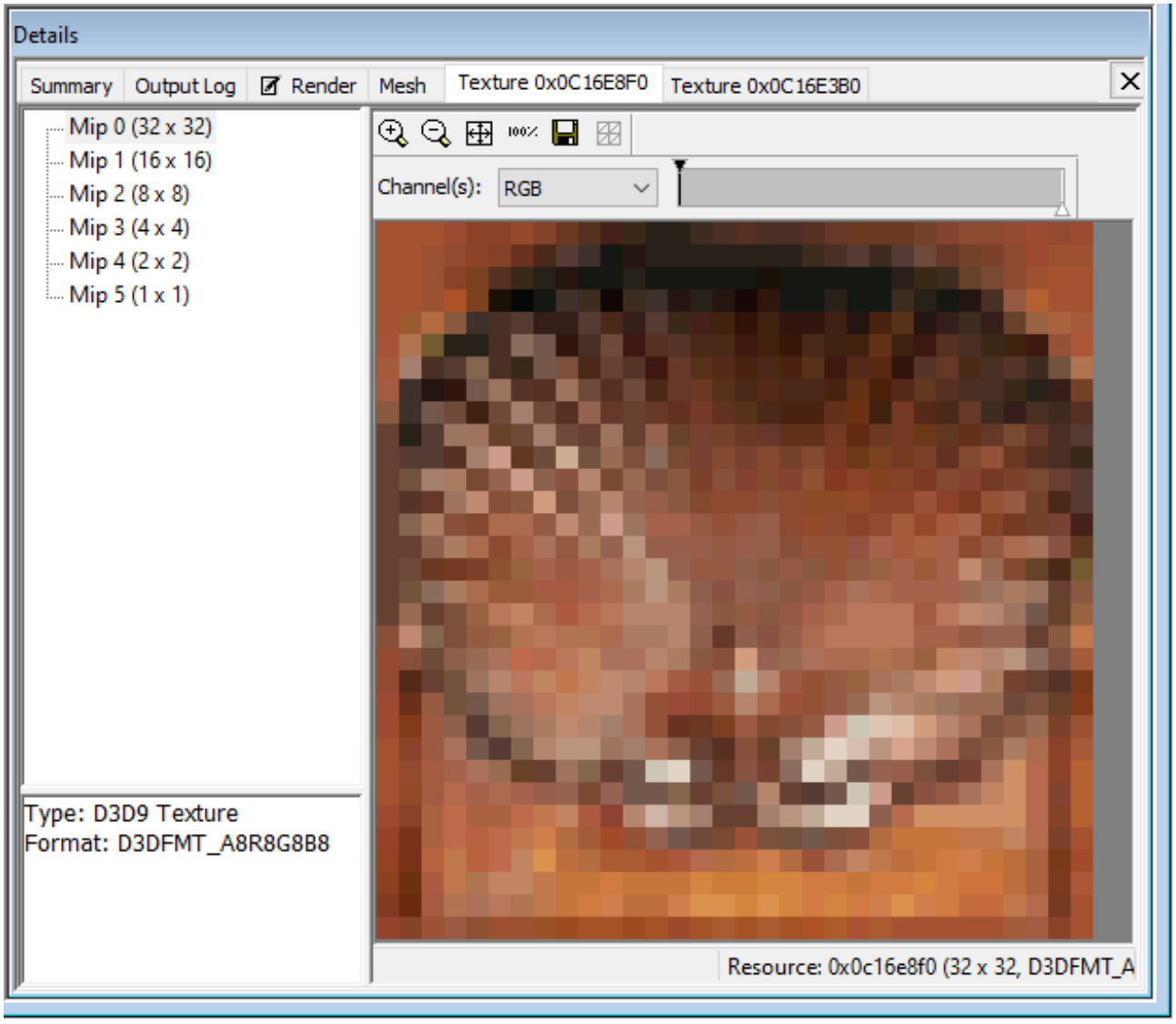


Figure D.2: Bed headboard texture

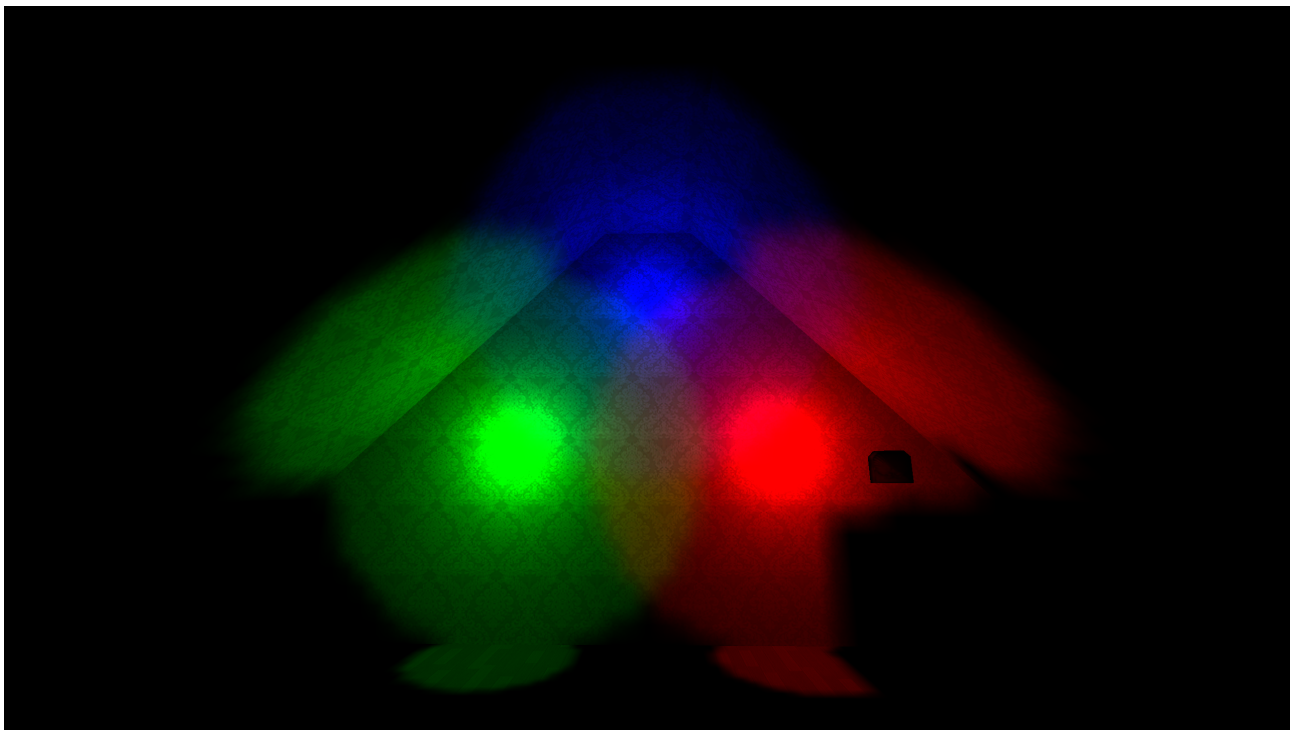
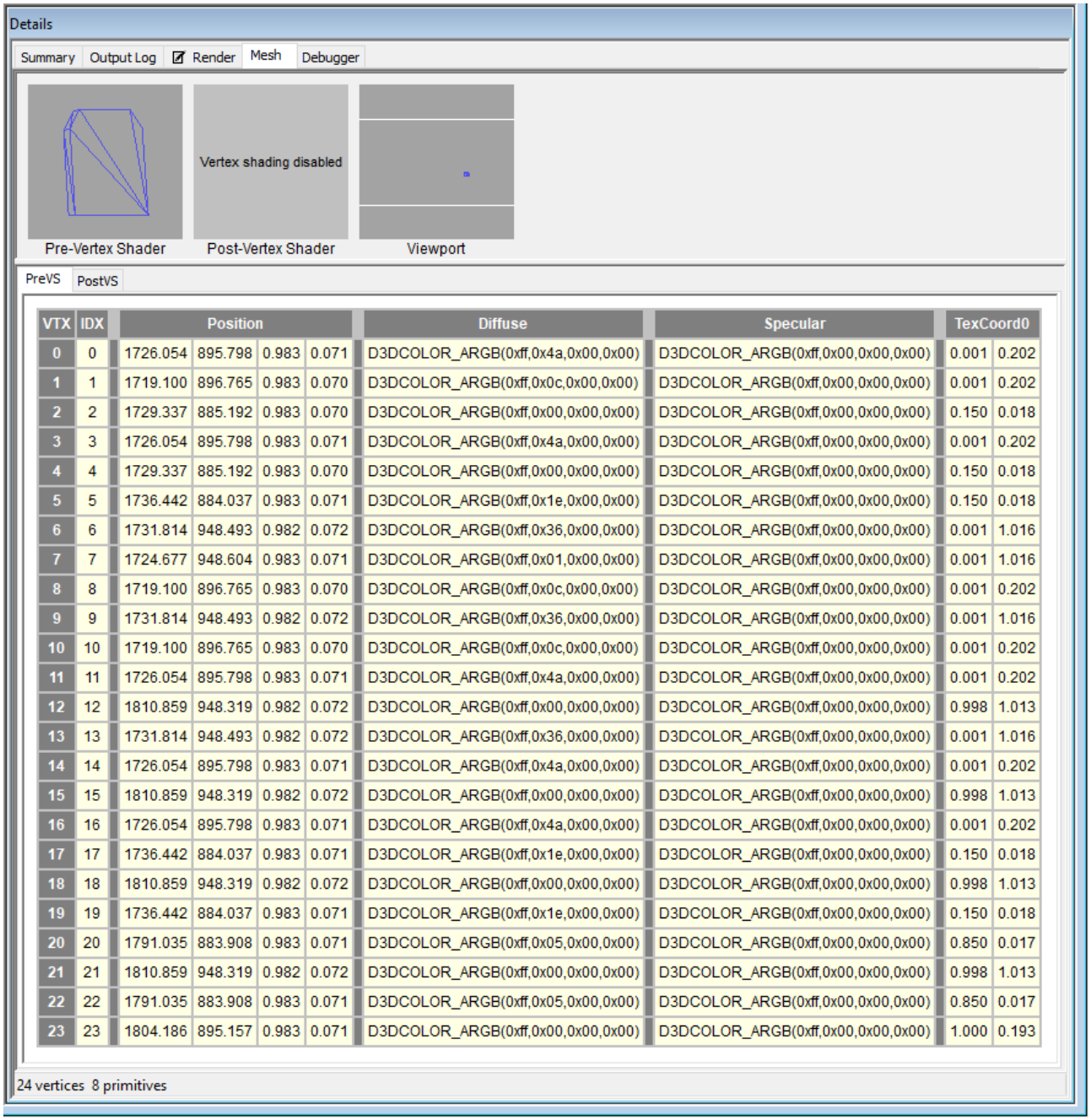


Figure D.3: Render of the first mesh with vertex lighting



VTX	IDX	Position				Diffuse	Specular	TexCoord0	
0	0	1726.054	895.798	0.983	0.071	D3DCOLOR_ARGB(0xff,0x4a,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
1	1	1719.100	896.765	0.983	0.070	D3DCOLOR_ARGB(0xff,0x0c,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
2	2	1729.337	885.192	0.983	0.070	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.150	0.018
3	3	1726.054	895.798	0.983	0.071	D3DCOLOR_ARGB(0xff,0x4a,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
4	4	1729.337	885.192	0.983	0.070	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.150	0.018
5	5	1736.442	884.037	0.983	0.071	D3DCOLOR_ARGB(0xff,0x1e,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.150	0.018
6	6	1731.814	948.493	0.982	0.072	D3DCOLOR_ARGB(0xff,0x36,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	1.016
7	7	1724.677	948.604	0.983	0.071	D3DCOLOR_ARGB(0xff,0x01,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	1.016
8	8	1719.100	896.765	0.983	0.070	D3DCOLOR_ARGB(0xff,0x0c,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
9	9	1731.814	948.493	0.982	0.072	D3DCOLOR_ARGB(0xff,0x36,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	1.016
10	10	1719.100	896.765	0.983	0.070	D3DCOLOR_ARGB(0xff,0x0c,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
11	11	1726.054	895.798	0.983	0.071	D3DCOLOR_ARGB(0xff,0x4a,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
12	12	1810.859	948.319	0.982	0.072	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.998	1.013
13	13	1731.814	948.493	0.982	0.072	D3DCOLOR_ARGB(0xff,0x36,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	1.016
14	14	1726.054	895.798	0.983	0.071	D3DCOLOR_ARGB(0xff,0x4a,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
15	15	1810.859	948.319	0.982	0.072	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.998	1.013
16	16	1726.054	895.798	0.983	0.071	D3DCOLOR_ARGB(0xff,0x4a,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.001	0.202
17	17	1736.442	884.037	0.983	0.071	D3DCOLOR_ARGB(0xff,0x1e,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.150	0.018
18	18	1810.859	948.319	0.982	0.072	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.998	1.013
19	19	1736.442	884.037	0.983	0.071	D3DCOLOR_ARGB(0xff,0x1e,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.150	0.018
20	20	1791.035	883.908	0.983	0.071	D3DCOLOR_ARGB(0xff,0x05,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.850	0.017
21	21	1810.859	948.319	0.982	0.072	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.998	1.013
22	22	1791.035	883.908	0.983	0.071	D3DCOLOR_ARGB(0xff,0x05,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	0.850	0.017
23	23	1804.186	895.157	0.983	0.071	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	D3DCOLOR_ARGB(0xff,0x00,0x00,0x00)	1.000	0.193

Figure D.4: Details of the first mesh with vertex lighting

E Lightmap and render - reg, green and blue torches lit

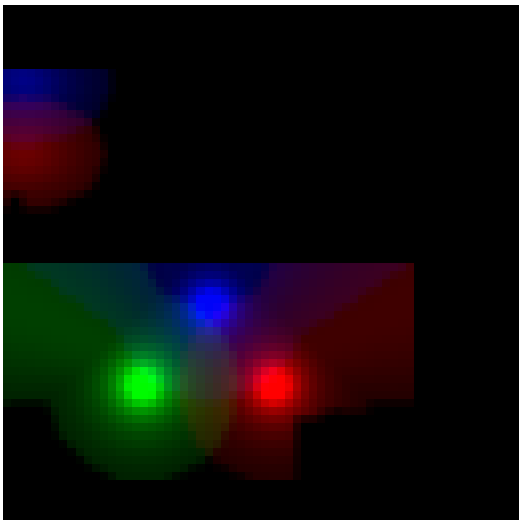


Figure E.1: Lightmap of scene with all torches lit

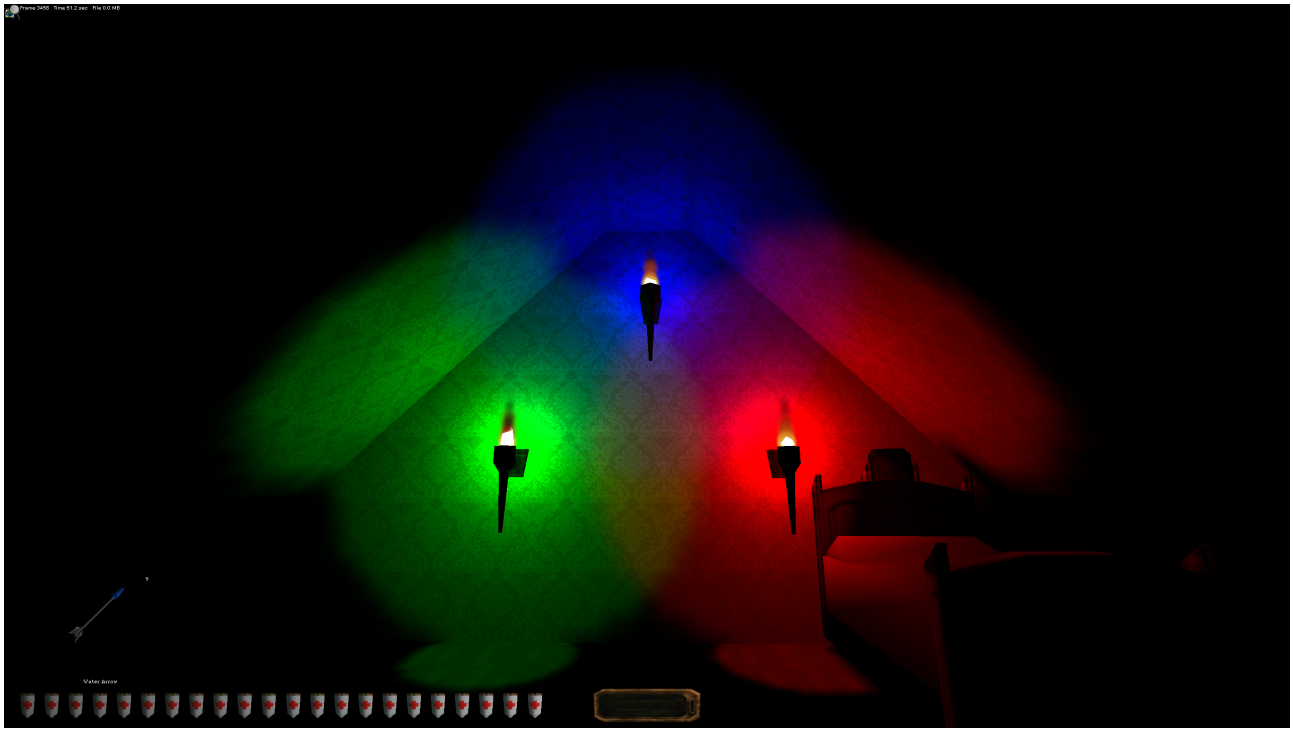


Figure E.2: Render of scene with all torches lit

F Lightmap and render - red and green torches lit

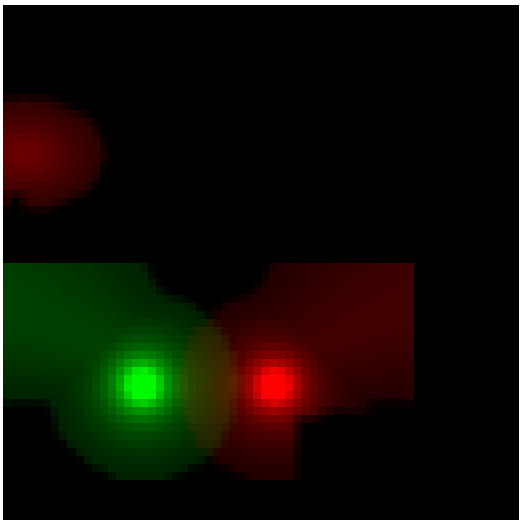


Figure F.1: Lightmap of scene with red and green torches lit

Figure F.2: Render of scene with red and green torches lit

G Lightmap and render - red and blue torches lit

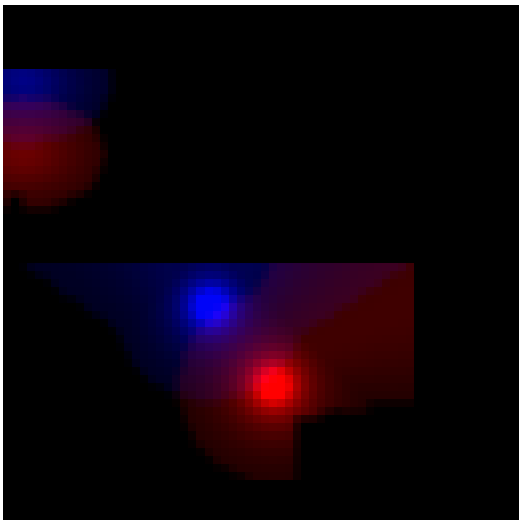


Figure G.1: Lightmap of scene with red and blue torches lit

Figure G.2: Render of scene with red and blue torches lit

H Lightmap and render - green and blue torches lit

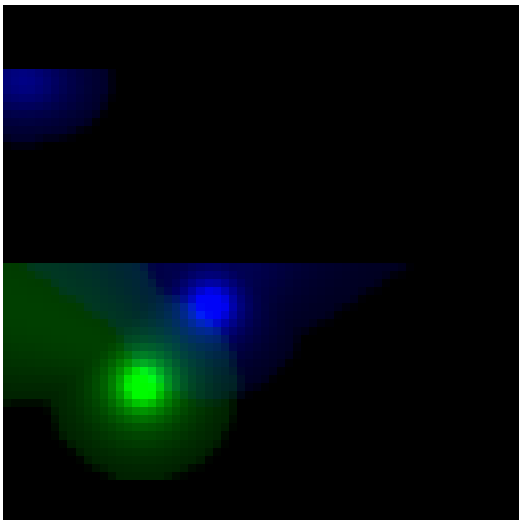


Figure H.1: Lightmap of scene with green and blue torches lit

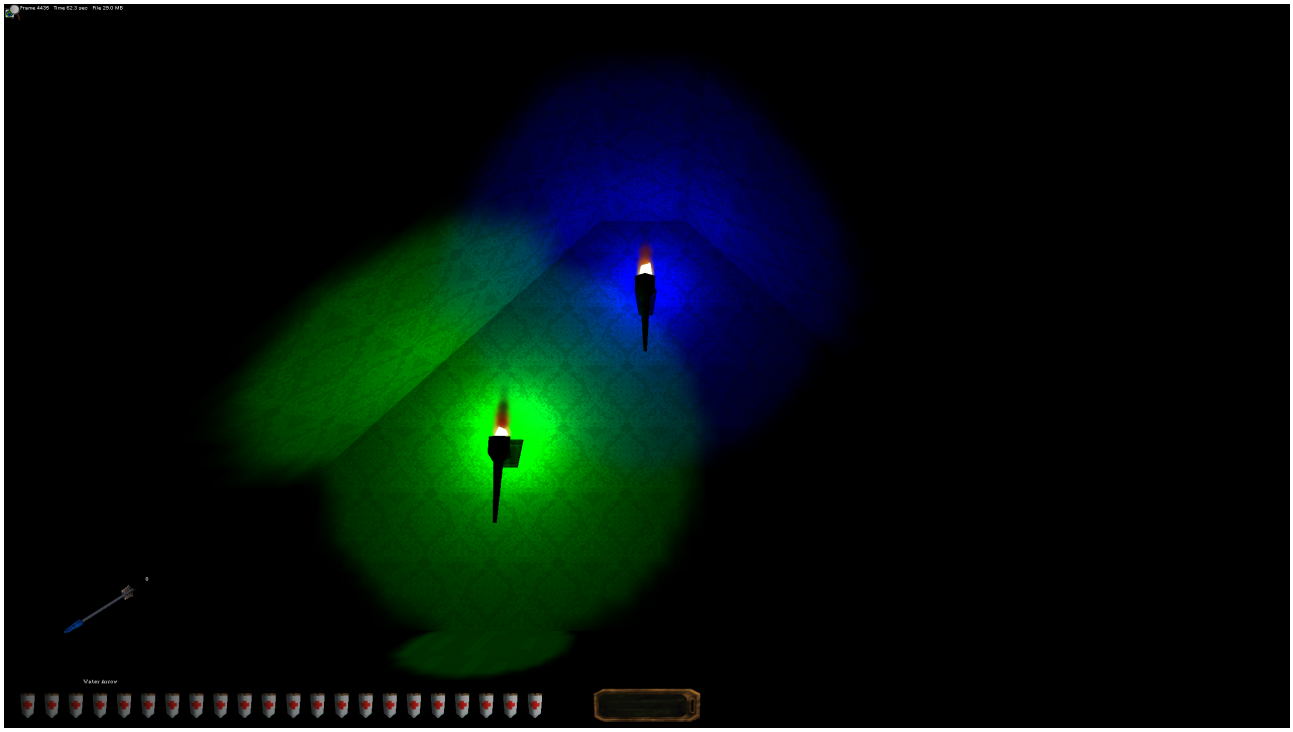


Figure H.2: Render of scene with green and blue torches lit

I Lightmap and render - red torch lit

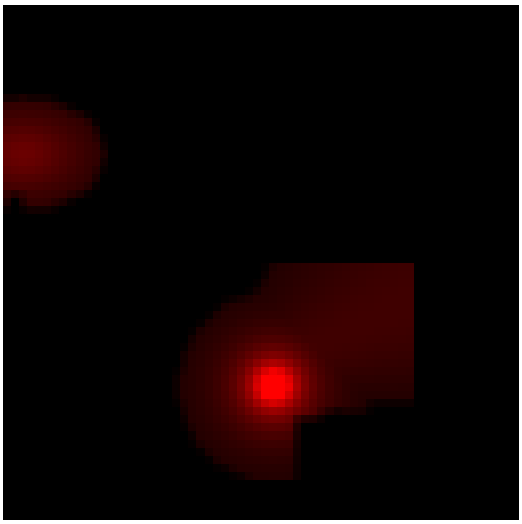


Figure I.1: Lightmap of scene with the red torch lit

Figure I.2: Render of scene with the red torch lit

J Lightmap and render - green torch lit

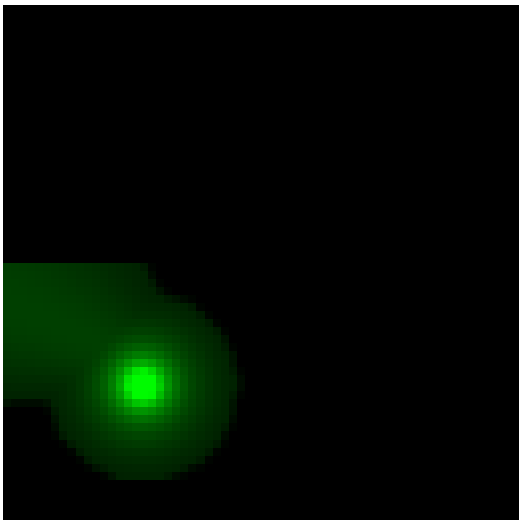


Figure J.1: Lightmap of scene with the green torch lit

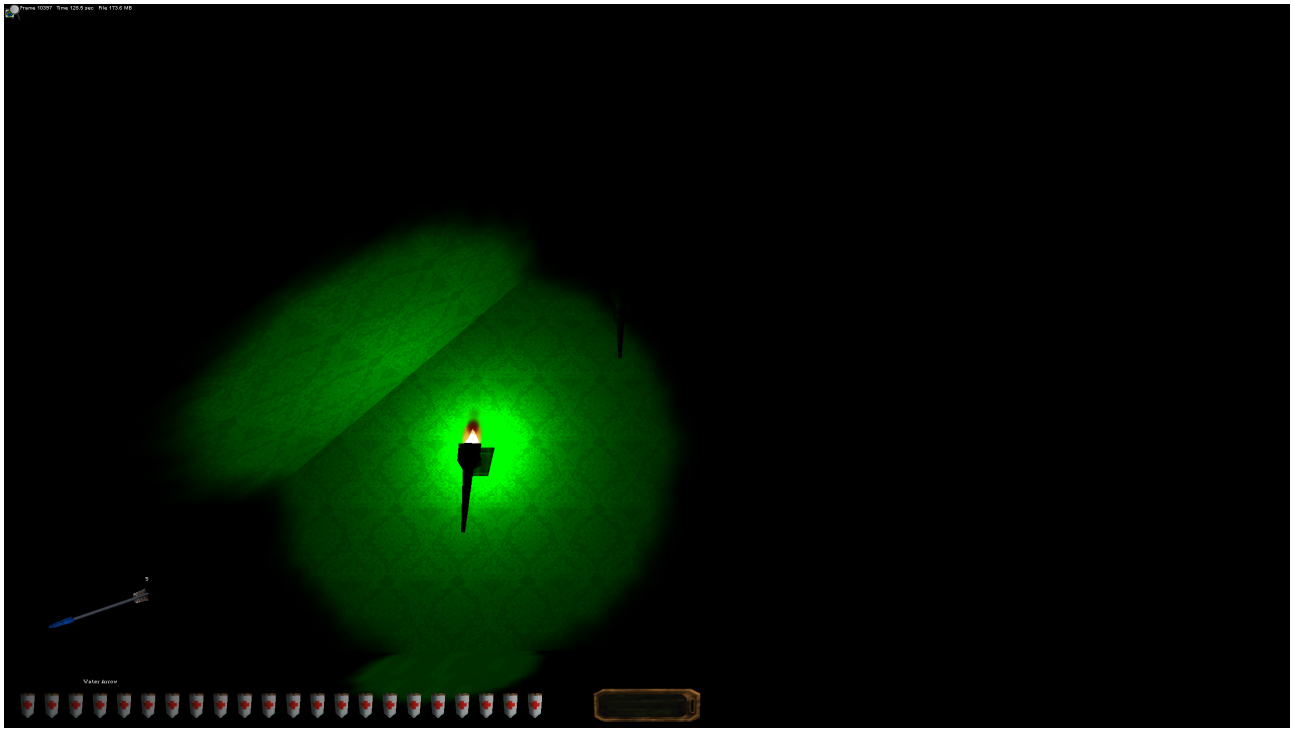


Figure J.2: Render of scene with the green torch lit

K Lightmap and render - blue torch lit

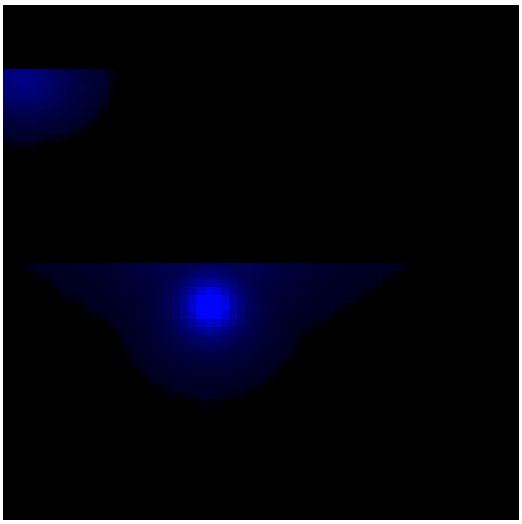


Figure K.1: Lightmap of scene with the blue torch lit

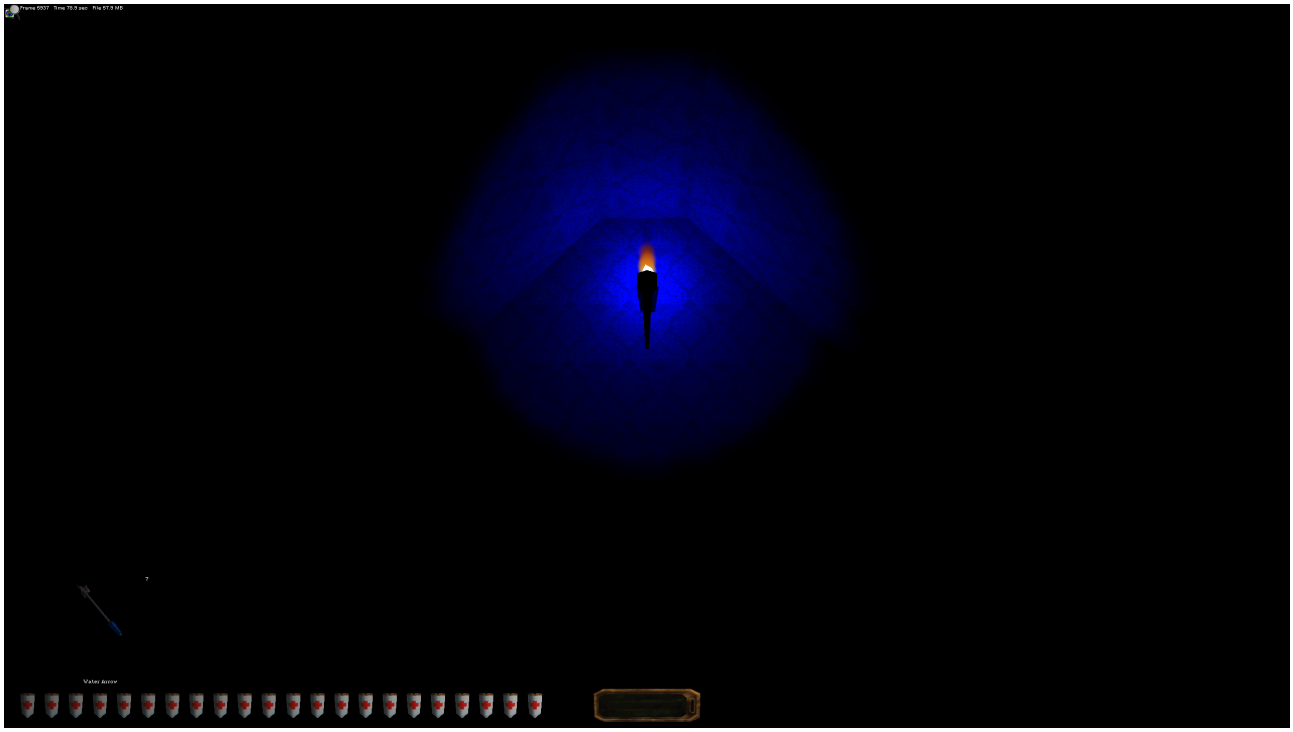


Figure K.2: Render of scene with the blue torch lit

L Lightmap and render - no torch lit

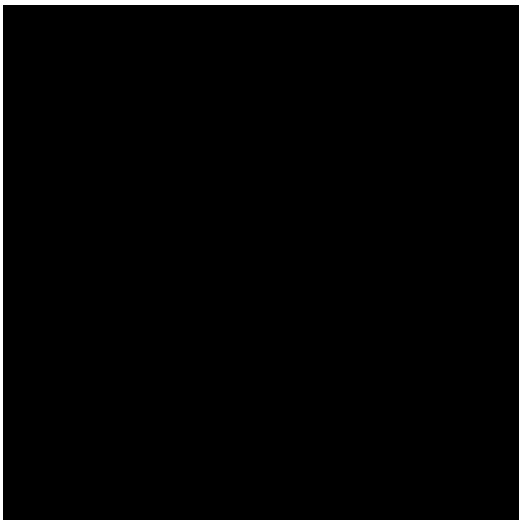


Figure L.1: Lightmap of scene with no torch lit

Figure L.2: Render of scene with no torch lit