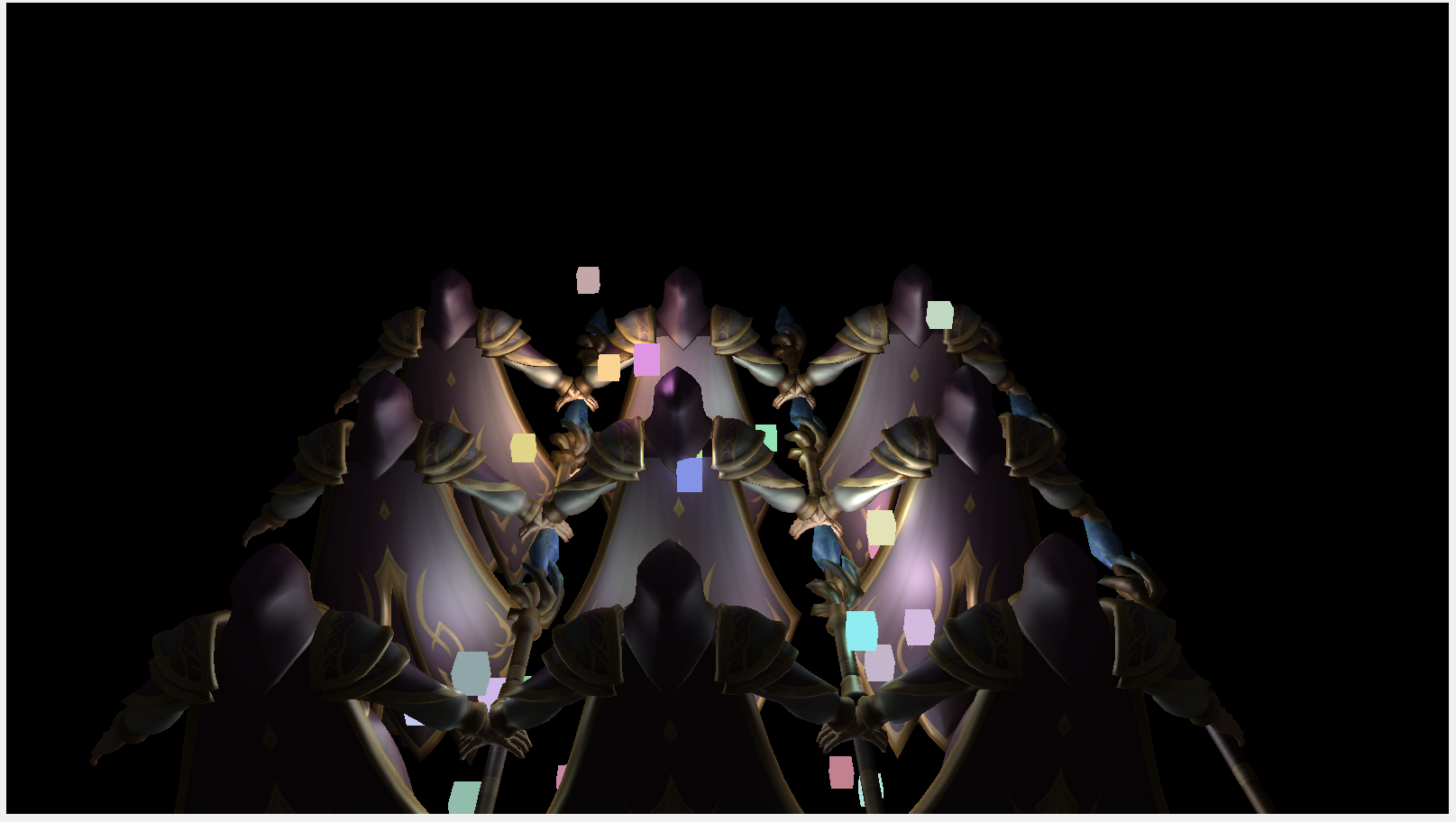
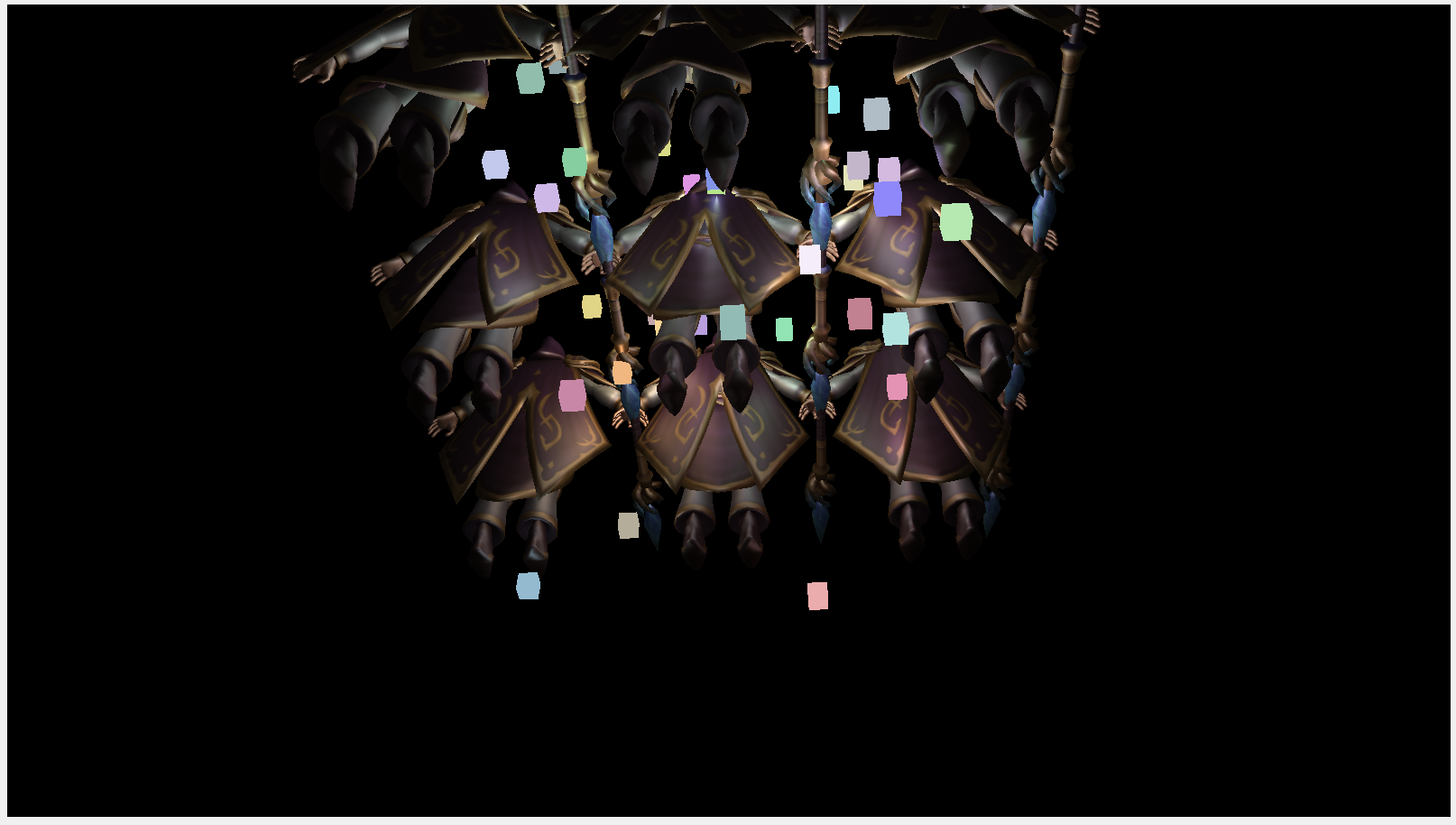
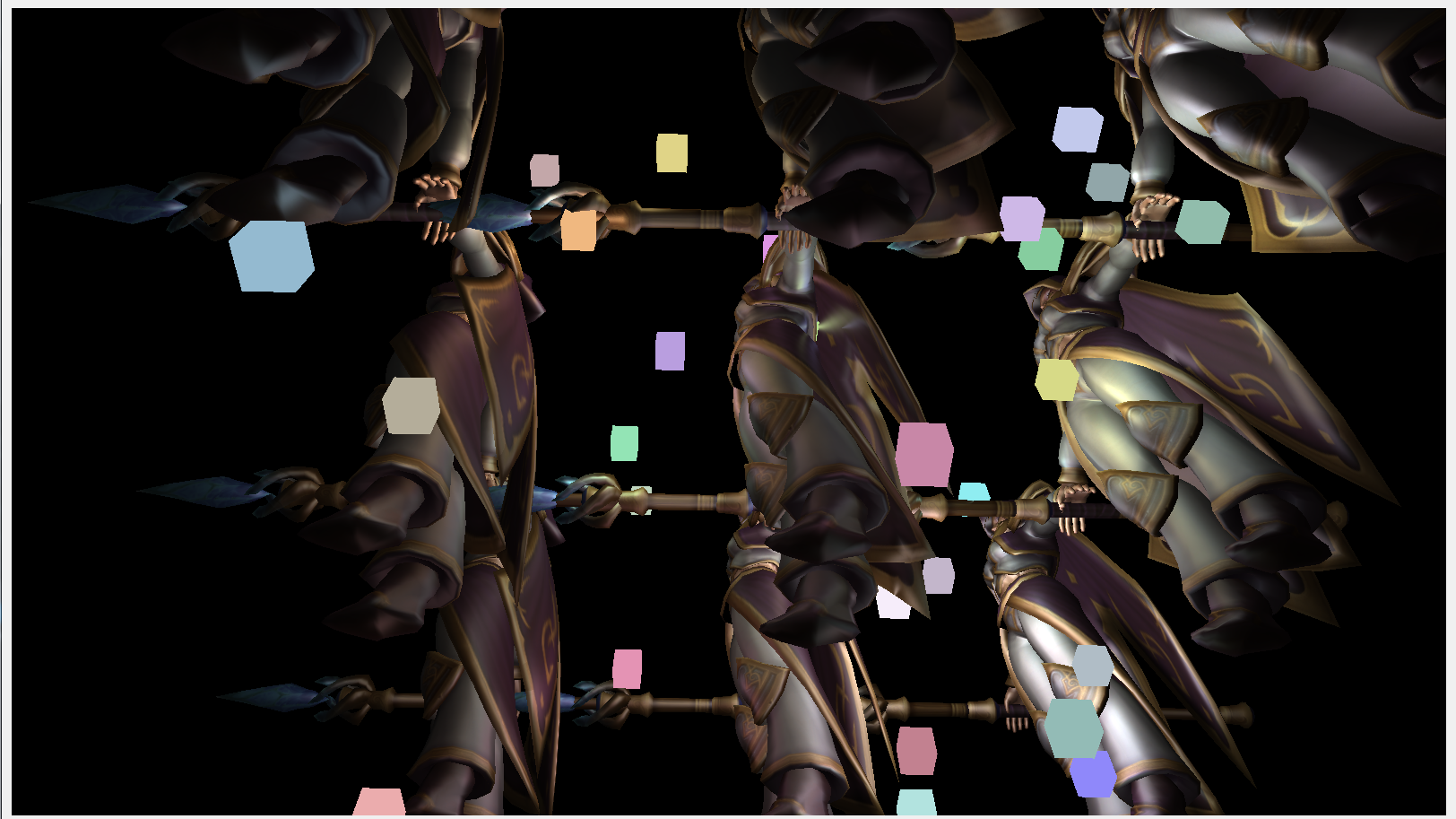
Lab 09 – Deferred Shading – Joshua Kauer

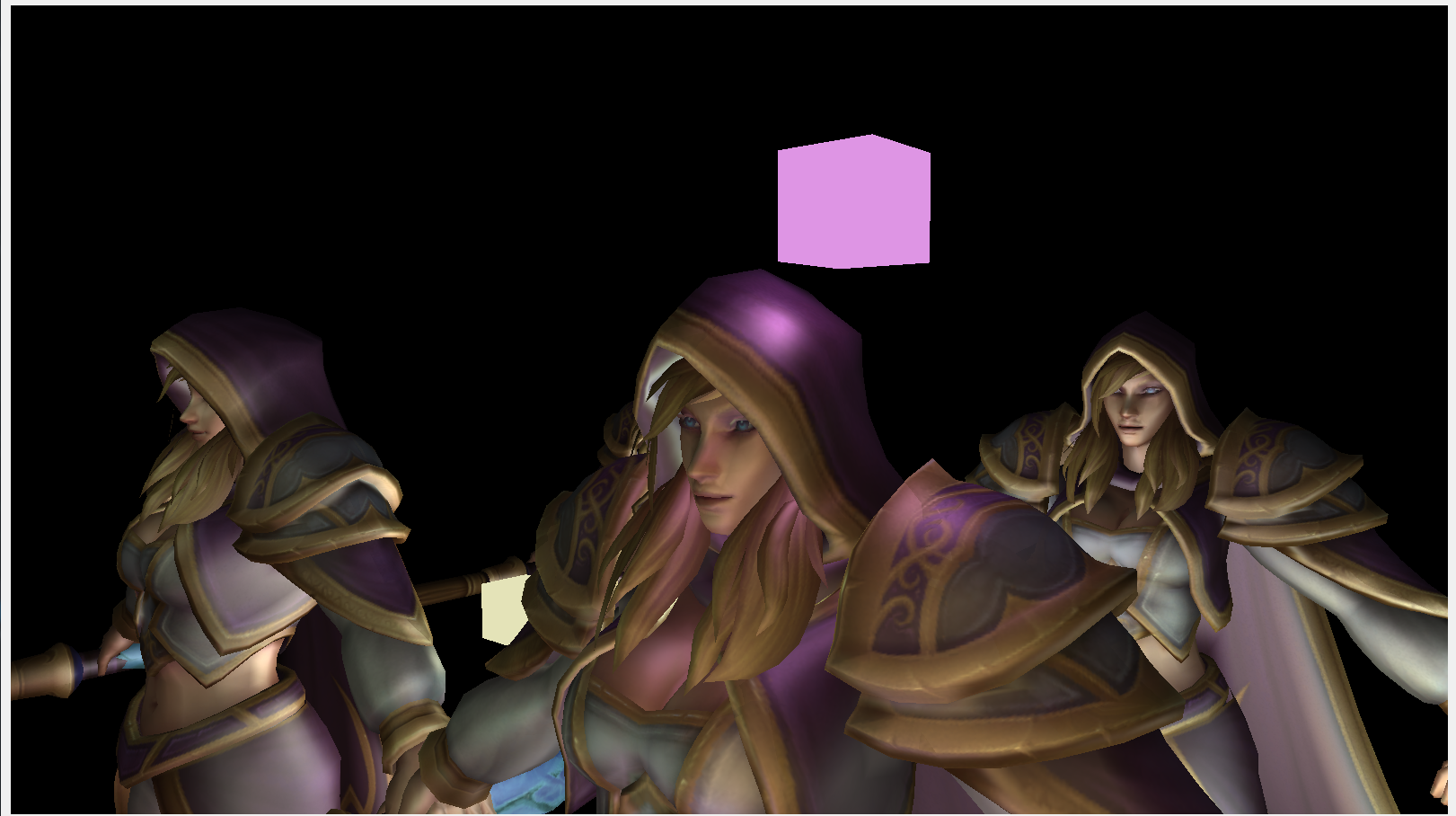
For this lab I did deferred shading which is based on the idea that I defer or postpone most of the heavy rendering (like lighting) to a later stage. Deferred shading consists of two passes: in the first pass called the geometry pass I render the scene once and retrieve all kinds of geometrical information from the objects that I store in a collection of textures called the G-buffer; think of position vectors, color vectors, normal vectors and/or specular values. The geometric information of a scene stored in the G-buffer is then later used for (more complex) lighting calculations. I use the textures from the G-buffer in a second pass where I render it onto a quad and calculate the scene's lighting for each fragment using the geometrical information stored in the G-buffer. The lighting calculates are exactly the same with forward rendering lighting only this time I take all required input variables from the corresponding G-buffer textures instead of the vertex shader.













//Geometry Pass Shader

//Vertex

#version 430

layout (location = 0) in vec3 position;

layout (location = 1) in vec2 texCoords;

layout (location = 2) in vec3 normal;

out vec3 FragPos;

out vec2 TexCoords;

out vec3 Normal;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main()

{

vec4 worldPos = model \* vec4(position, 1.0f);

FragPos = worldPos.xyz;

gl\_Position = projection \* view \* worldPos;

TexCoords = texCoords;

mat3 normalMatrix = transpose(inverse(mat3(model)));

Normal = normalMatrix \* normal;

}

//Fragment

#version 430

layout (location = 0) out vec3 gPosition;

layout (location = 1) out vec3 gNormal;

layout (location = 2) out vec4 gAlbedoSpec;

in vec2 TexCoords;

in vec3 FragPos;

in vec3 Normal;

uniform sampler2D texture\_diffuse1;

uniform sampler2D texture\_specular1;

void main()

{

// Store the fragment position vector in the first gbuffer texture

gPosition = FragPos;

// Also store the per-fragment normals into the gbuffer

gNormal = normalize(Normal);

// And the diffuse per-fragment color

gAlbedoSpec.rgb = texture(texture\_diffuse1, TexCoords).rgb;

// Store specular intensity in gAlbedoSpec's alpha component

gAlbedoSpec.a = texture(texture\_specular1, TexCoords).r;

}

//Lighting Pass Shader

//Vertex

#version 430

layout (location = 0) in vec3 position;

layout (location = 1) in vec2 texCoords;

out vec2 TexCoords;

void main()

{

gl\_Position = vec4(position, 1.0f);

TexCoords = texCoords;

}

//Fragment

#version 430

out vec4 FragColor;

in vec2 TexCoords;

uniform sampler2D gPosition;

uniform sampler2D gNormal;

uniform sampler2D gAlbedoSpec;

struct Light {

vec3 Position;

vec3 Color;

float Linear;

float Quadratic;

};

const int NR\_LIGHTS = 32;

uniform Light lights[NR\_LIGHTS];

uniform vec3 viewPos;

void main()

{

// Retrieve data from gbuffer

vec3 FragPos = texture(gPosition, TexCoords).rgb;

vec3 Normal = texture(gNormal, TexCoords).rgb;

vec3 Diffuse = texture(gAlbedoSpec, TexCoords).rgb;

float Specular = texture(gAlbedoSpec, TexCoords).a;

// Then calculate lighting as usual

vec3 lighting = Diffuse \* 0.1; // hard-coded ambient component

vec3 viewDir = normalize(viewPos - FragPos);

for(int i = 0; i < NR\_LIGHTS; ++i)

{

// Diffuse

vec3 lightDir = normalize(lights[i].Position - FragPos);

vec3 diffuse = max(dot(Normal, lightDir), 0.0) \* Diffuse \* lights[i].Color;

// Specular

vec3 halfwayDir = normalize(lightDir + viewDir);

float spec = pow(max(dot(Normal, halfwayDir), 0.0), 16.0);

vec3 specular = lights[i].Color \* spec \* Specular;

// Attenuation

float distance = length(lights[i].Position - FragPos);

float attenuation = 1.0 / (1.0 + lights[i].Linear \* distance + lights[i].Quadratic \* distance \* distance);

diffuse \*= attenuation;

specular \*= attenuation;

lighting += diffuse + specular;

}

FragColor = vec4(lighting, 1.0);

}

//Light Cube Shader

//Vertex

#version 430

layout (location = 0) in vec3 position;

layout (location = 1) in vec3 normal;

layout (location = 2) in vec2 texCoords;

uniform mat4 projection;

uniform mat4 view;

uniform mat4 model;

void main()

{

gl\_Position = projection \* view \* model \* vec4(position, 1.0f);

}

//Fragment

#version 430

layout (location = 0) out vec4 FragColor;

uniform vec3 lightColor;

void main()

{

FragColor = vec4(lightColor, 1.0);

}