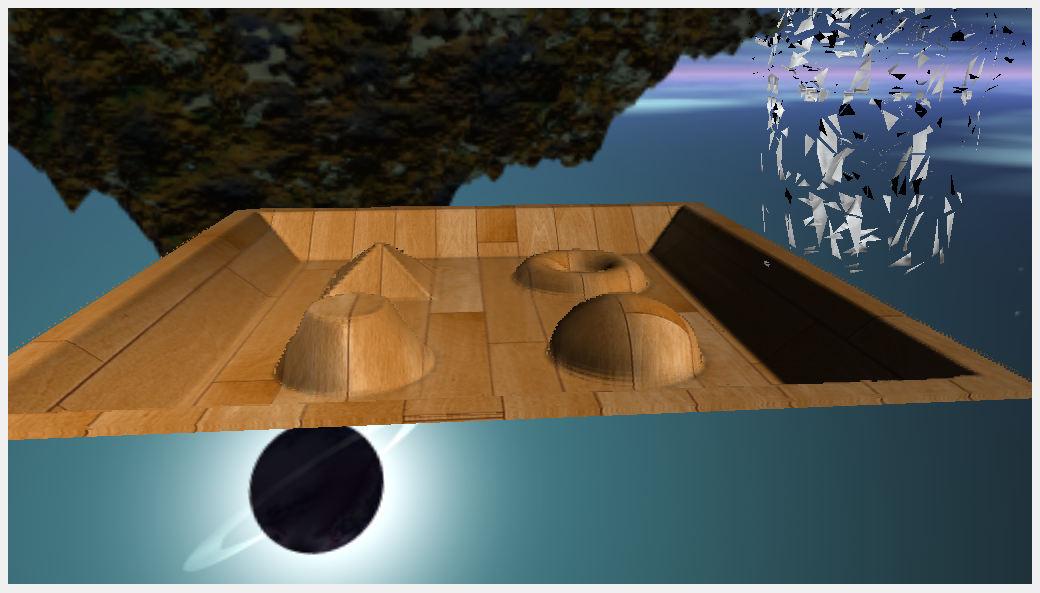
Lab 08 – Parallax Mapping – Joshua Kauer

In this lab, I extended my work on Normal Mapping and did parallax mapping which will add even more depth to a simple plane. This is done by adding a height map that has black and white values, and will then modify how it looks like on the screen. You can even look at it from the side and it won’t lose its depth.





//Vertex Shader

#version 430

layout (location = 0) in vec3 position;

layout (location = 1) in vec2 texCoords;

layout (location = 2) in vec3 normal;

layout (location = 3) in vec3 tangent;

layout (location = 4) in vec3 bitangent;

out VS\_OUT {

vec3 FragPos;

vec2 TexCoords;

vec3 TangentLightPos;

vec3 TangentViewPos;

vec3 TangentFragPos;

vec3 Normal;

vec3 Tangent;

vec3 Bitangent;

} vs\_out;

uniform mat4 projection;

uniform mat4 view;

uniform mat4 model;

uniform vec3 lightPos;

uniform vec3 viewPos;

void main()

{

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

vs\_out.FragPos = vec3(model \* vec4(position, 1.0));

vs\_out.TexCoords = texCoords;

vs\_out.Normal = normal;

vs\_out.Tangent = tangent;

vs\_out.Bitangent = bitangent;

vec3 T = normalize(mat3(model) \* tangent);

vec3 B = normalize(mat3(model) \* bitangent);

vec3 N = normalize(mat3(model) \* normal);

mat3 TBN = transpose(mat3(T, B, N));

vs\_out.TangentLightPos = TBN \* lightPos;

vs\_out.TangentViewPos = TBN \* viewPos;

vs\_out.TangentFragPos = TBN \* vs\_out.FragPos;

}

//Fragment Shader

#version 430

out vec4 FragColor;

in VS\_OUT {

vec3 FragPos;

vec2 TexCoords;

vec3 TangentLightPos;

vec3 TangentViewPos;

vec3 TangentFragPos;

vec3 Normal;

vec3 Tangent;

vec3 Bitangent;

} fs\_in;

uniform sampler2D diffuseMap;

uniform sampler2D normalMap;

uniform sampler2D depthMap;

uniform float height\_scale;

vec2 ParallaxMapping(vec2 texCoords, vec3 viewDir)

{

// number of depth layers

const float minLayers = 10;

const float maxLayers = 20;

float numLayers = mix(maxLayers, minLayers, abs(dot(vec3(0.0, 0.0, 1.0), viewDir)));

// calculate the size of each layer

float layerDepth = 1.0 / numLayers;

// depth of current layer

float currentLayerDepth = 0.0;

// the amount to shift the texture coordinates per layer (from vector P)

vec2 P = viewDir.xy / viewDir.z \* height\_scale;

vec2 deltaTexCoords = P / numLayers;

// get initial values

vec2 currentTexCoords = texCoords;

float currentDepthMapValue = texture(depthMap, currentTexCoords).r;

while(currentLayerDepth < currentDepthMapValue)

{

// shift texture coordinates along direction of P

currentTexCoords -= deltaTexCoords;

// get depthmap value at current texture coordinates

currentDepthMapValue = texture(depthMap, currentTexCoords).r;

// get depth of next layer

currentLayerDepth += layerDepth;

}

// -- parallax occlusion mapping interpolation from here on

// get texture coordinates before collision (reverse operations)

vec2 prevTexCoords = currentTexCoords + deltaTexCoords;

// get depth after and before collision for linear interpolation

float afterDepth = currentDepthMapValue - currentLayerDepth;

float beforeDepth = texture(depthMap, prevTexCoords).r - currentLayerDepth + layerDepth;

// interpolation of texture coordinates

float weight = afterDepth / (afterDepth - beforeDepth);

vec2 finalTexCoords = prevTexCoords \* weight + currentTexCoords \* (1.0 - weight);

return finalTexCoords;

}

void main()

{

// Offset texture coordinates with Parallax Mapping

vec3 viewDir = normalize(fs\_in.TangentViewPos - fs\_in.TangentFragPos);

vec2 texCoords = fs\_in.TexCoords;

texCoords = ParallaxMapping(fs\_in.TexCoords, viewDir);

// discards a fragment when sampling outside default texture region (fixes border artifacts)

if(texCoords.x > 1.0 || texCoords.y > 1.0 || texCoords.x < 0.0 || texCoords.y < 0.0)

discard;

// Obtain normal from normal map

vec3 normal = texture(normalMap, texCoords).rgb;

normal = normalize(normal \* 2.0 - 1.0);

// Get diffuse color

vec3 color = texture(diffuseMap, texCoords).rgb;

// Ambient

vec3 ambient = 0.1 \* color;

// Diffuse

vec3 lightDir = normalize(fs\_in.TangentLightPos - fs\_in.TangentFragPos);

float diff = max(dot(lightDir, normal), 0.0);

vec3 diffuse = diff \* color;

// Specular

vec3 reflectDir = reflect(-lightDir, normal);

vec3 halfwayDir = normalize(lightDir + viewDir);

float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);

vec3 specular = vec3(0.2) \* spec;

FragColor = vec4(ambient + diffuse + specular, 1.0f);

}