UNIVERSITÉ Concordia UNIVERSITY

COMP 371

Computer Graphics

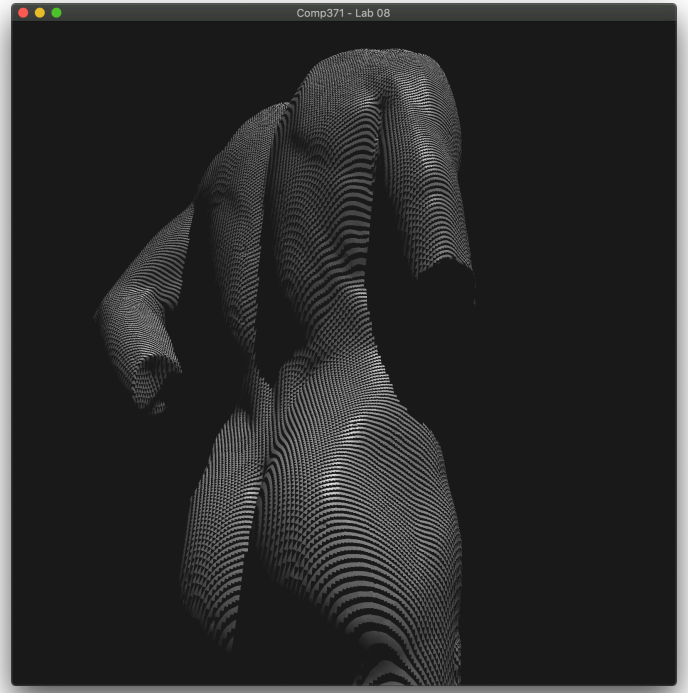
Lab 08

Shadow Mapping

Prepared by Simon Demeule

Graphics from learnopengl

Expected results



Shadow Acne

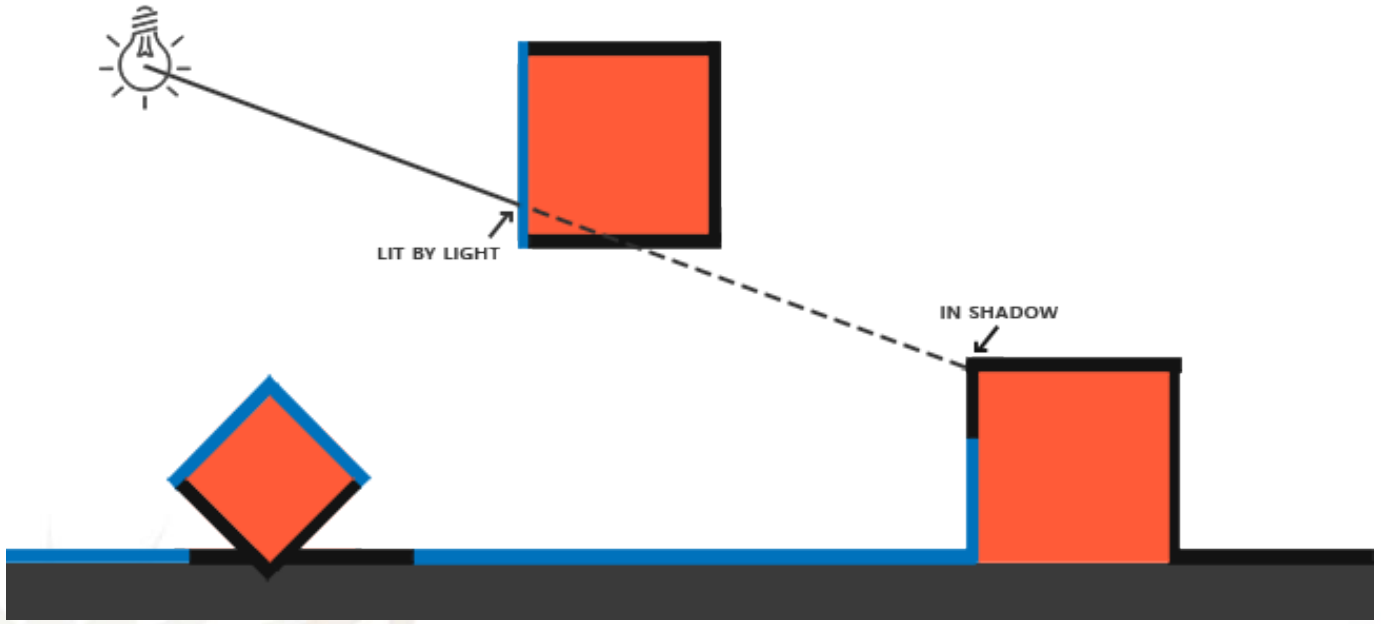
Shadow Acne Removed

Install instructions

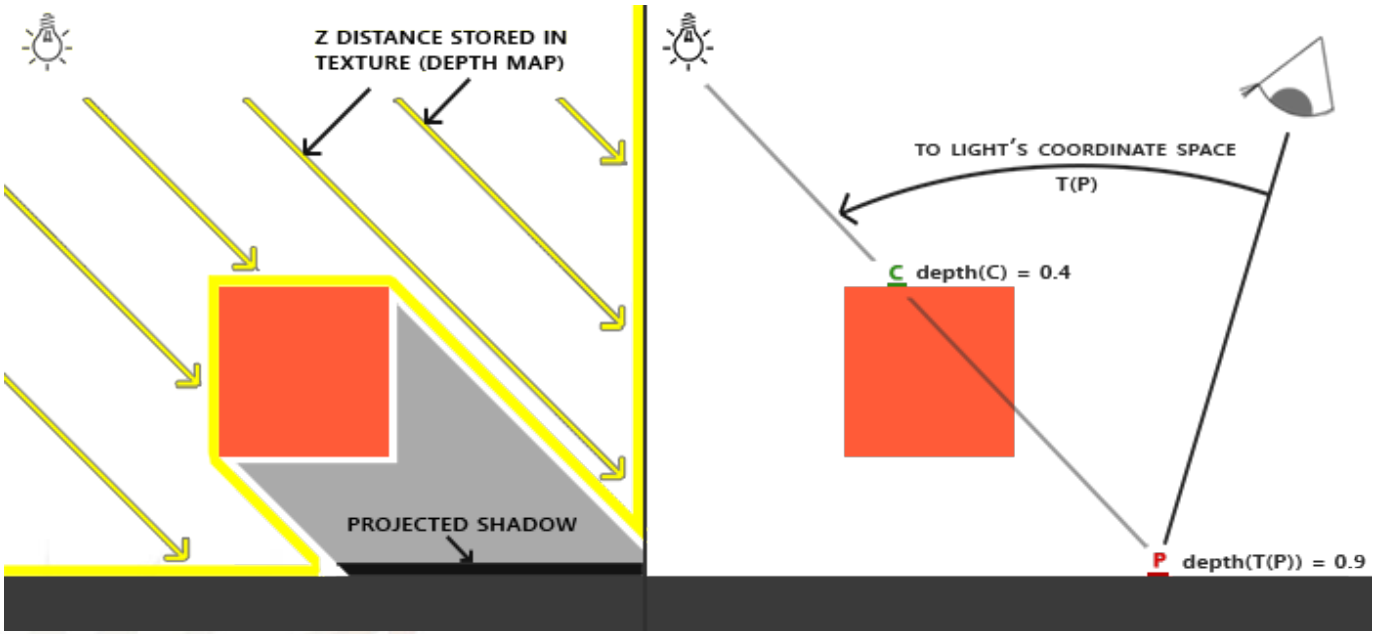
Get Lab08.zip on Moodle.
The lab is standalone, no need to add into previous lab code.

Shadow Mapping Algorithm

- Create shadows cast by a point light source



1. Place a camera at the light's position and see how close the nearest object is for every angle
2. Decide wether a surface is in shadow by comparing it's distance to the light with the closest distance at that angle



How do we do step 1? (getting the closest distance at each angle)

1. Render a depth map onto a texture that shaders can then access
2. Must create a projection and view matrix that define where the light is looking at
3. Must create a framebuffer that allows opengl to draw onto the texture

Creating the texture and framebuffer

```
// Setup texture and framebuffer for creating shadow map

// Variable storing index to framebuffer used for shadow mapping
GLuint depthMapFBO;
// Get the framebuffer
glGenFramebuffers(1, &depthMapFBO);
// Dimensions of the shadow texture
const unsigned int SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
// Variable storing index to texture used for shadow mapping
GLuint depthMap;
// Get the texture
glGenTextures(1, &depthMap);
// Bind the texture so the next glTex calls affect it
glBindTexture(GL_TEXTURE_2D, depthMap);
// Create the texture and specify it's attributes, including widthn height, components (only depth is stored, no color information)
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
// Set texture sampler parameters.
// The two calls below tell the texture sampler inside the shader how to upsample and downsample the texture. Here we choose the
    nearest filtering option, which means we just use the value of the closest pixel to the chosen image coordinate.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
// The two calls below tell the texture sampler inside the shader how it should deal with texture coordinates outside of the [0, 1]
    range. Here we decide to just tile the image.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// Bind the framebuffer so the next glFramebuffer calls affect it
glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
// Attach the depth map texture to the depth map framebuffer
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, depthMap, 0);

// Set texture unit # for shadow map
setShadowMapTexture(shaderShadow, 0);
```

Placing the light and giving it a projection matrix

```
// Calculate variables for shadow mapping
vec3 lightPosition(0.0, 20.0, 10.0);   // the location of the light in 3D space
vec3 lightFocus(0.0, 0.0, 0.0);        // the point in 3D space the light "looks" at
vec3 lightDirection = normalize(lightFocus - lightPosition);

float lightAngleOuter = 30.0;
float lightAngleInner = 20.0;
float lightNearPlane = 1.0f;
float lightFarPlane = 80.0f;

mat4 lightProjectionMatrix = perspective(20.0f, (float)SHADOW_WIDTH / (float)SHADOW_HEIGHT, lightNearPlane, lightFarPlane);
mat4 lightViewMatrix = lookAt(lightPosition, lightFocus, vec3(0.0f, 0.0f, 1.0f));
mat4 lightSpaceMatrix = lightProjectionMatrix * lightViewMatrix;
```

The shadow shader
all we care about is depth

vertex

```
#version 330 core
layout (location = 0) in vec3 position;

uniform mat4 light_space_matrix;
uniform mat4 model_matrix;

void main()
{
    gl_Position = light_space_matrix * model_matrix * vec4(position, 1.0);
}
```

fragment

```
#version 330 core

void main()
{
    gl_FragDepth = gl_FragCoord.z;
}
```

How do we do step 2? (deciding if a surface is in the shadow or not)

1. Compute the distance between the 3D position of the fragment and the light
2. Use the light's projection matrix to see where the 3D position of the fragments ends up on the depth texture
3. Compare the computed distance to the one stored on the depth texture.

   computed > depth texture => in shadow

Render the geometry as usual

```
// Render scene

// Use proper shader
glUseProgram(shaderScene);
// Use proper image output size
// Side note: we get the size from the framebuffer instead of using WIDTH and HEIGHT because of a bug with highDPI displays
int width, height;
glfwGetFramebufferSize(window, &width, &height);
glViewport(0, 0, width, height);
// Bind screen as output framebuffer
glBindFramebuffer(GL_FRAMEBUFFER, 0);
// Clear color and depth data on framebuffer
glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Bind depth map texture
glActiveTexture(GL_TEXTURE0);
// Bind geometry
glBindVertexArray(activeVAO);
// Draw geometry
glDrawElements(GL_TRIANGLES, activeVertices, GL_UNSIGNED_INT, 0);
// Unbind geometry
glBindVertexArray(0);
```

The scene shader
where the magic happens

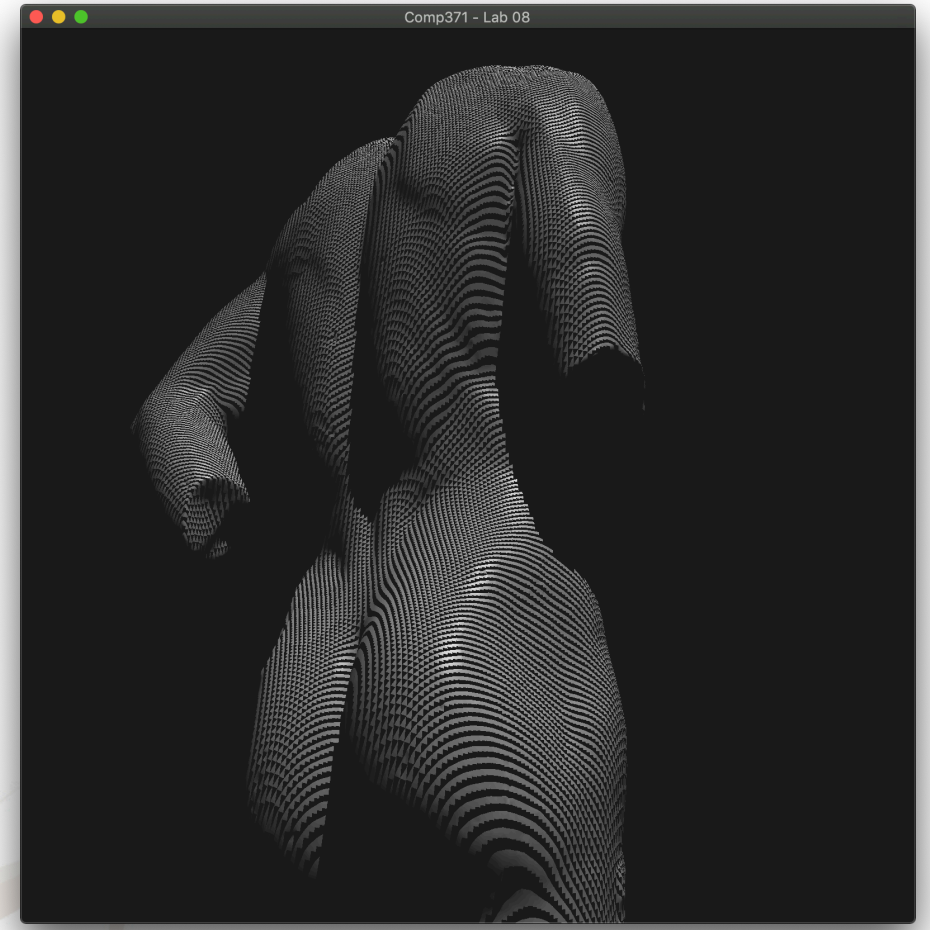
vertex

```
void main()
{
    fragment_normal = mat3(model_matrix) * normals;
    fragment_position = vec3(model_matrix * vec4(position, 1.0));
    fragment_position_light_space = light_space_matrix * vec4(fragment_position, 1.0);
    gl_Position = projection_matrix * view_matrix * model_matrix * vec4(position, 1.0);
}
```

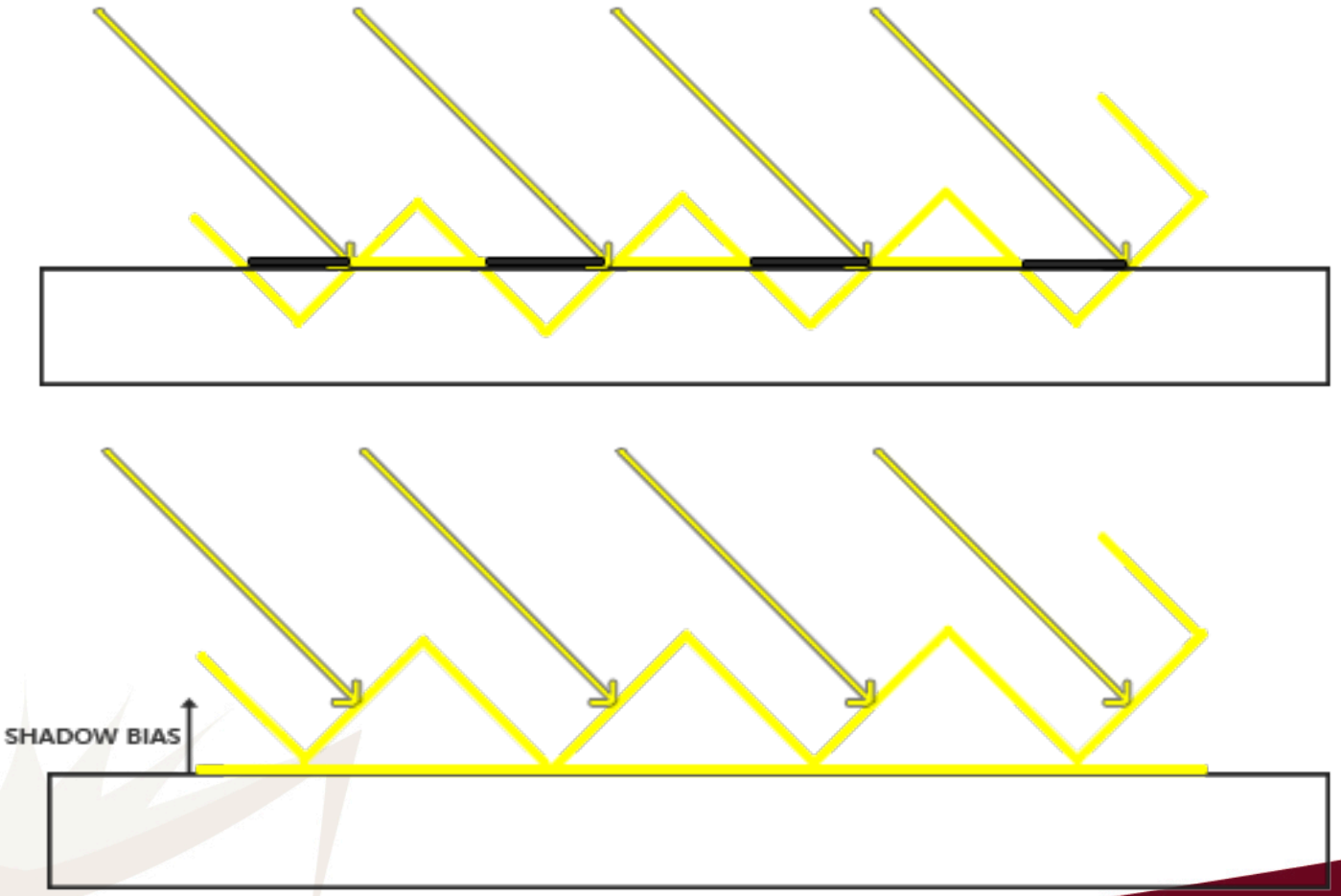
fragment

```
float shadow_scalar() {
    // this function returns 1.0 when the surface recieves light, and 0.0 when it is in a shadow
    // perform perspective divide
    vec3 projected_coordinates = fragment_position_light_space.xyz / fragment_position_light_space.w;
    // transform to [0,1] range
    projected_coordinates = projected_coordinates * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragment_position_light_space as coords)
    float closest_depth = texture(shadow_map, projected_coordinates.xy).r;
    // get depth of current fragment from light's perspective
    float current_depth = projected_coordinates.z;
    // check whether current frag pos is in shadow
    float bias = 0;
    return ((current_depth - bias) < closest_depth) ? 1.0 : 0.0;
}
```

Problem: shadow acne



Solution: shadow bias



Solution: shadow bias

```
float shadow_scalar() {
    // this function returns 1.0 when the surface recieves light, and 0.0 when it is in a shadow
    // perform perspective divide
    vec3 projected_coordinates = fragment_position_light_space.xyz / fragment_position_light_space.w;
    // transform to [0,1] range
    projected_coordinates = projected_coordinates * 0.5 + 0.5;
    // get closest depth value from light's perspective (using [0,1] range fragment_position_light_space as coords)
    float closest_depth = texture(shadow_map, projected_coordinates.xy).r;
    // get depth of current fragment from light's perspective
    float current_depth = projected_coordinates.z;
    // check whether current frag pos is in shadow
    float bias = 0.003;
    return ((current_depth - bias) < closest_depth) ? 1.0 : 0.0;
}
```

Solution: shadow bias