CSC 4356 / ME 4573 Interactive Computer Graphics

Shadow Mapping

Shadow mapping

The intuition behind shadow mapping is simple: render the scene to an FBO from viewpoint of the light source.

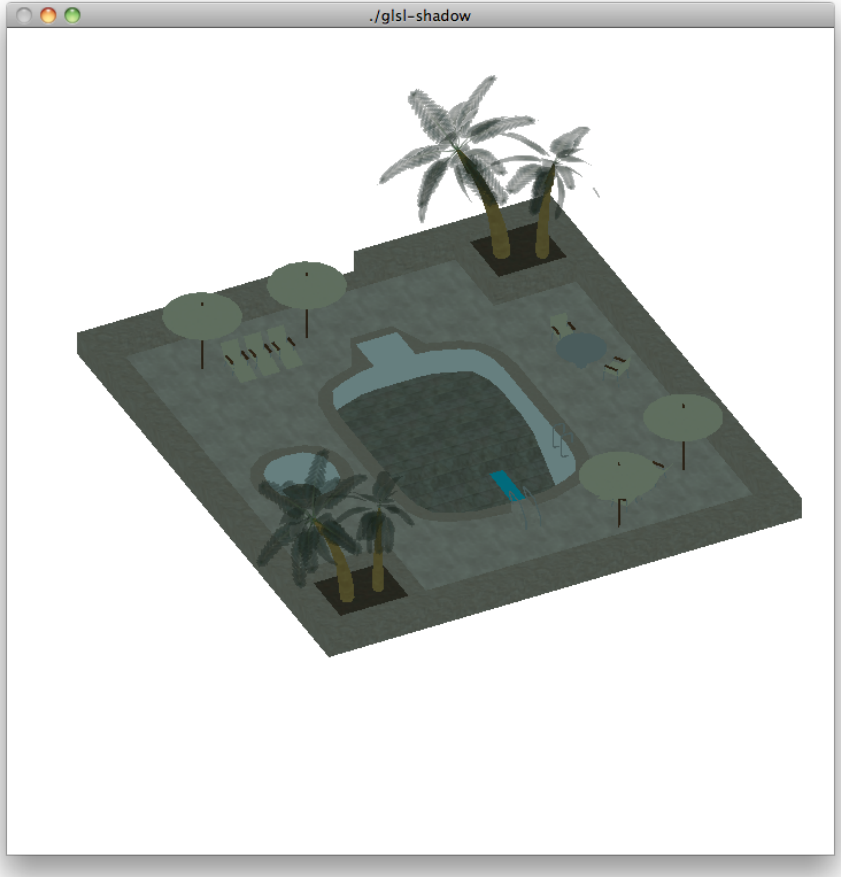
This buffer will tell us what the light can and cannot see.

This tells as what is and is not in shadow.

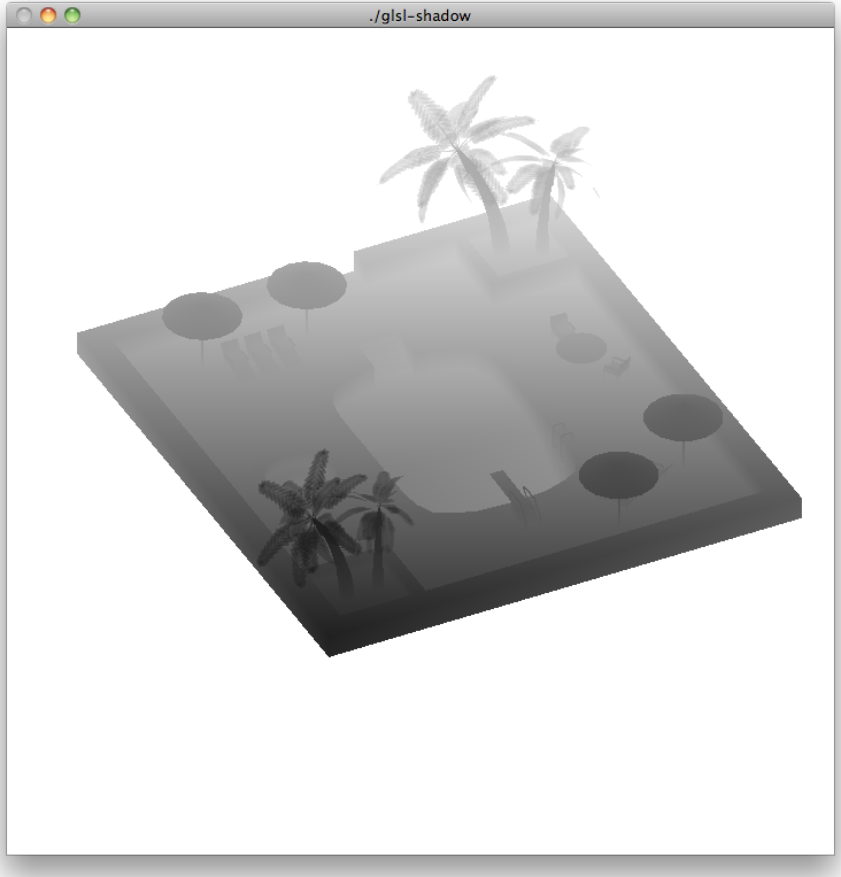
Shadow mapping

Shadow mapping is a *huge* topic in real-time 3D. This lecture covers only the very basics.

See *Real-Time Rendering* [Möller *et al*] for all the variations. Coverage and bibliography is excellent.



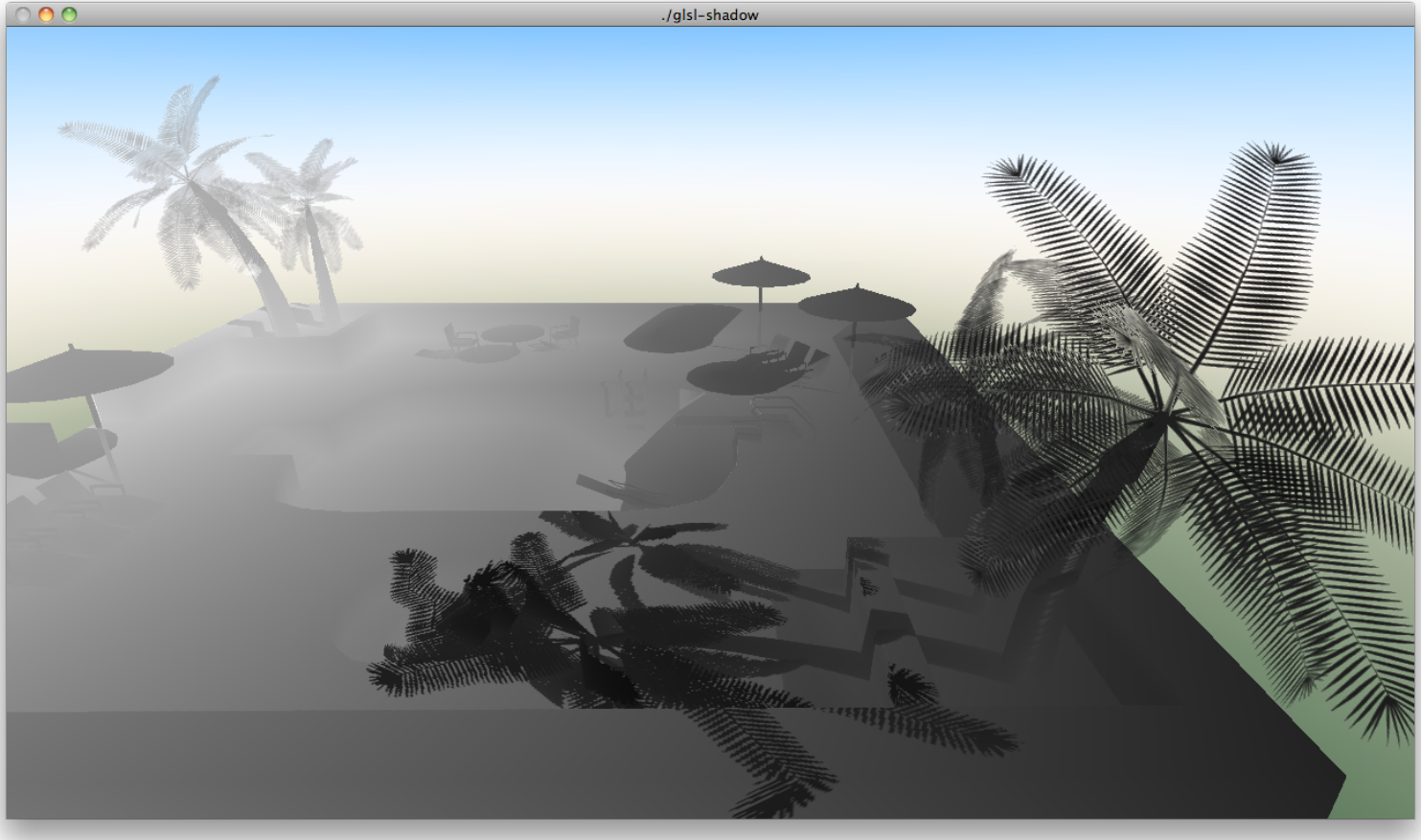
The color buffer, as viewed by the light source.



The depth buffer, as viewed by the light source.

A real-world projected texture.

A real-world projected texture.



Light source depth buffer, projectively textured.

Texture Projection

```
uniform sampler2D image;
```

Consider a 2D sampler uniform. A basic texture reference uses only the x and y values of the texture coordinate.

```
vec4 c = texture2D(image, gl_TexCoord[0].xy);
```

But a projected texture reference uses all four.

```
vec4 c = texture2DProj(image, gl_TexCoord[0].xyzw);
```

Texture Projection

This texture coordinate projection is *exactly* like vertex position projection.

Just as in vertex rasterization, the homegeneous divide occurs behind-the-scenes.

We can even build our projection matrix using ordinary OpenGL matrix generators: `glRotate`, `glOrtho`, etc.

```
GLfloat S[16] = {
    0.5f, 0.0f, 0.0f, 0.0f,
    0.0f, 0.5f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.5f, 0.0f,
    0.5f, 0.5f, 0.5f, 1.0f,
};
GLdouble size = 16.0;

glMatrixMode(GL_TEXTURE);
{
    glLoadMatrixf(S);
    glOrtho(-size, +size, -size, +size, -size, +size);
    glRotated(-light_x, 1.0, 0.0, 0.0);
    glRotated(-light_y, 0.0, 1.0, 0.0);
}
glMatrixMode(GL_MODELVIEW);
```

The shadow coordinate for each vertex is thus *generated*: the world-space vertex position transform by the light source matrix.

```
void main()
{
    gl_TexCoord[1] = gl_TextureMatrix[4] * gl_Vertex; // Shadow
    gl_TexCoord[0] = gl_MultiTexCoord0;               // Texture
    gl_Position    = ftransform();
}
```

Shadow map Z test

We now have two light source depth values to consider in the fragment shader:

1. The generated shadow coordinate z value
2. The rendered shadow map depth value

If these two values match, then the fragment is visible to the light source, and is thus illuminated.

Otherwise, that fragment is in shadow.

Shadow map Z test

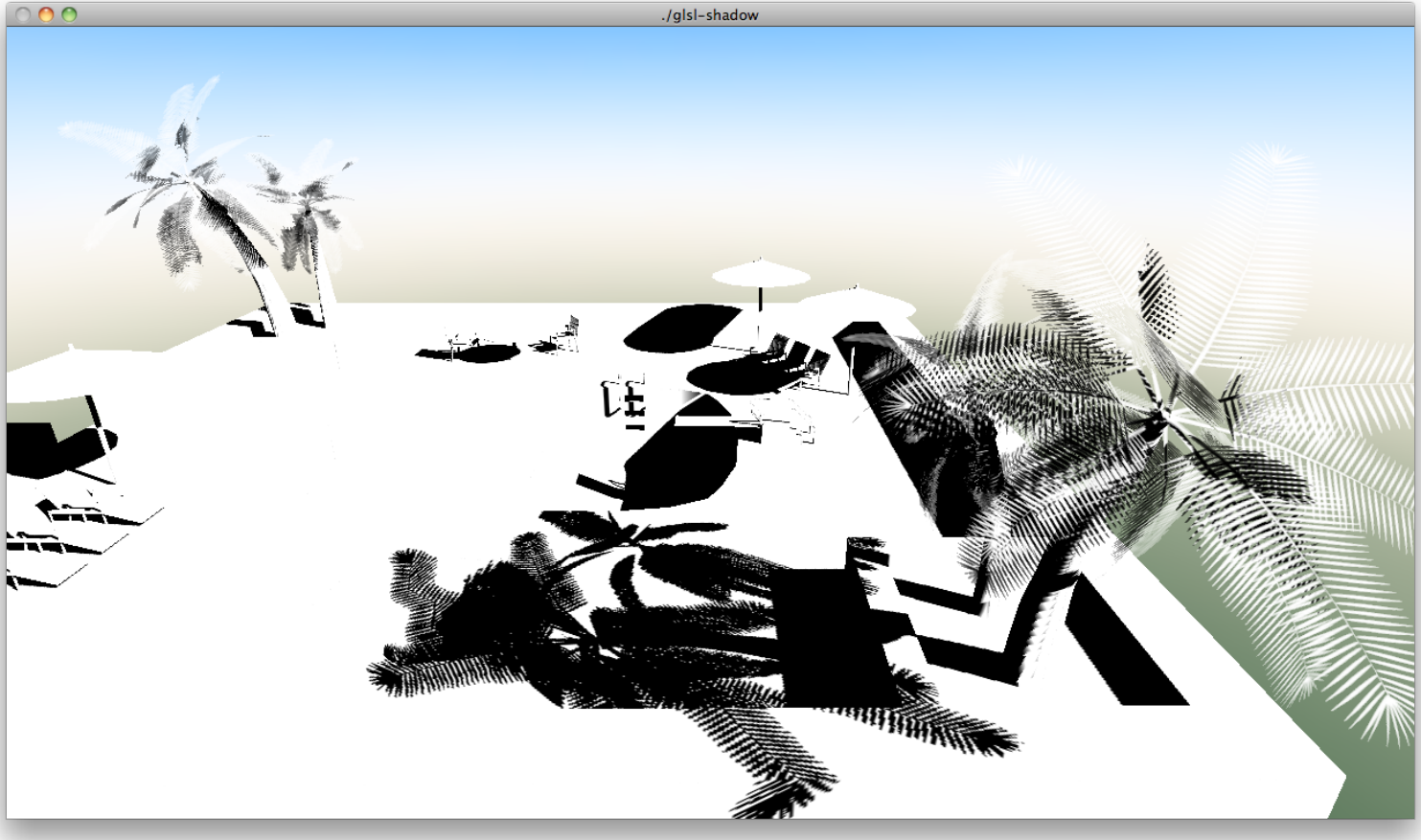
GLSL allows us to enable the comparison between texture coordinate z and depth map z using a texture parameter…

```
glBindTexture(GL_TEXTURE_2D, depth);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE_ARB,
                               GL_COMPARE_R_TO_TEXTURE_ARB);
```

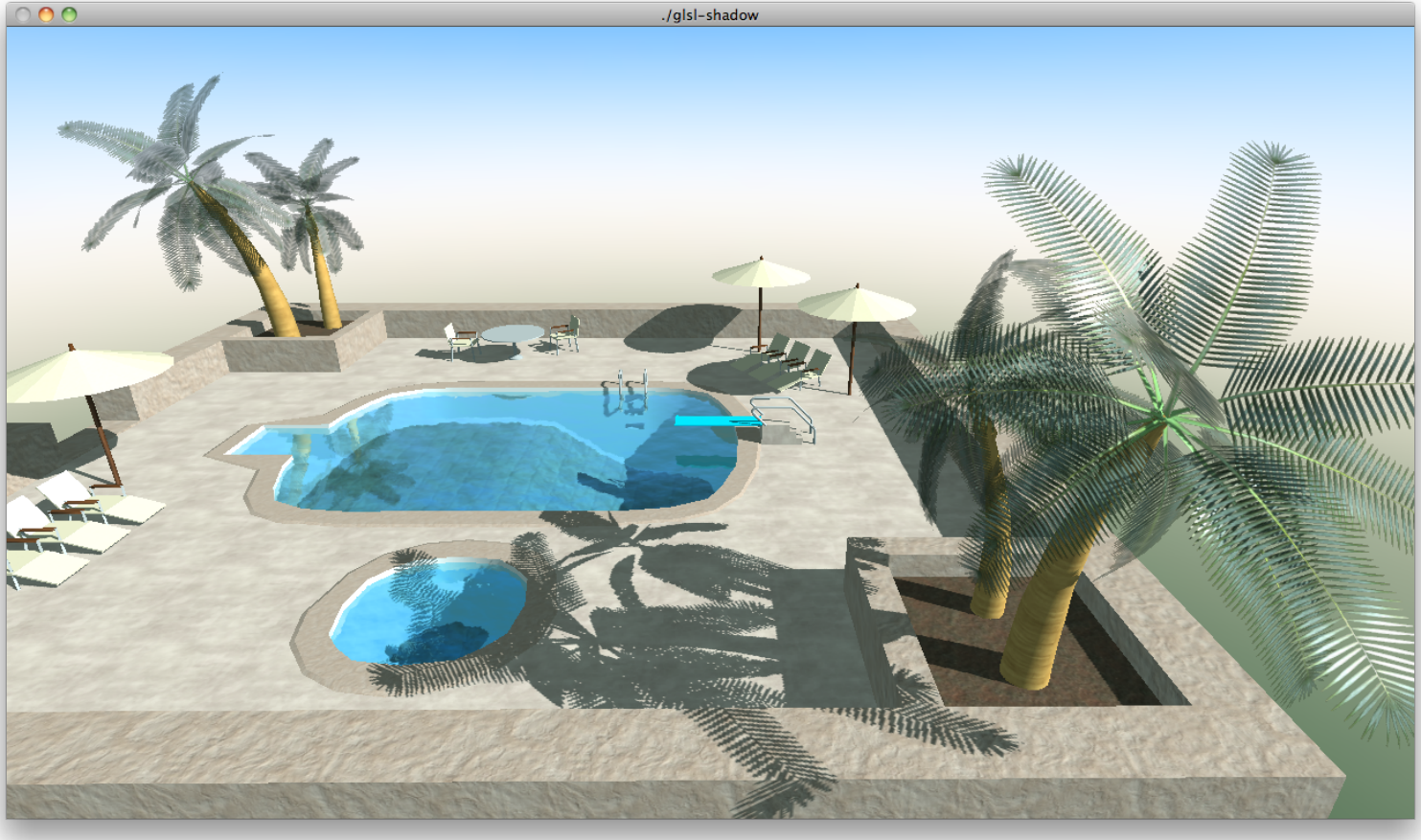
Shadow map Z test

The simplest possible shadow mapping fragment program:

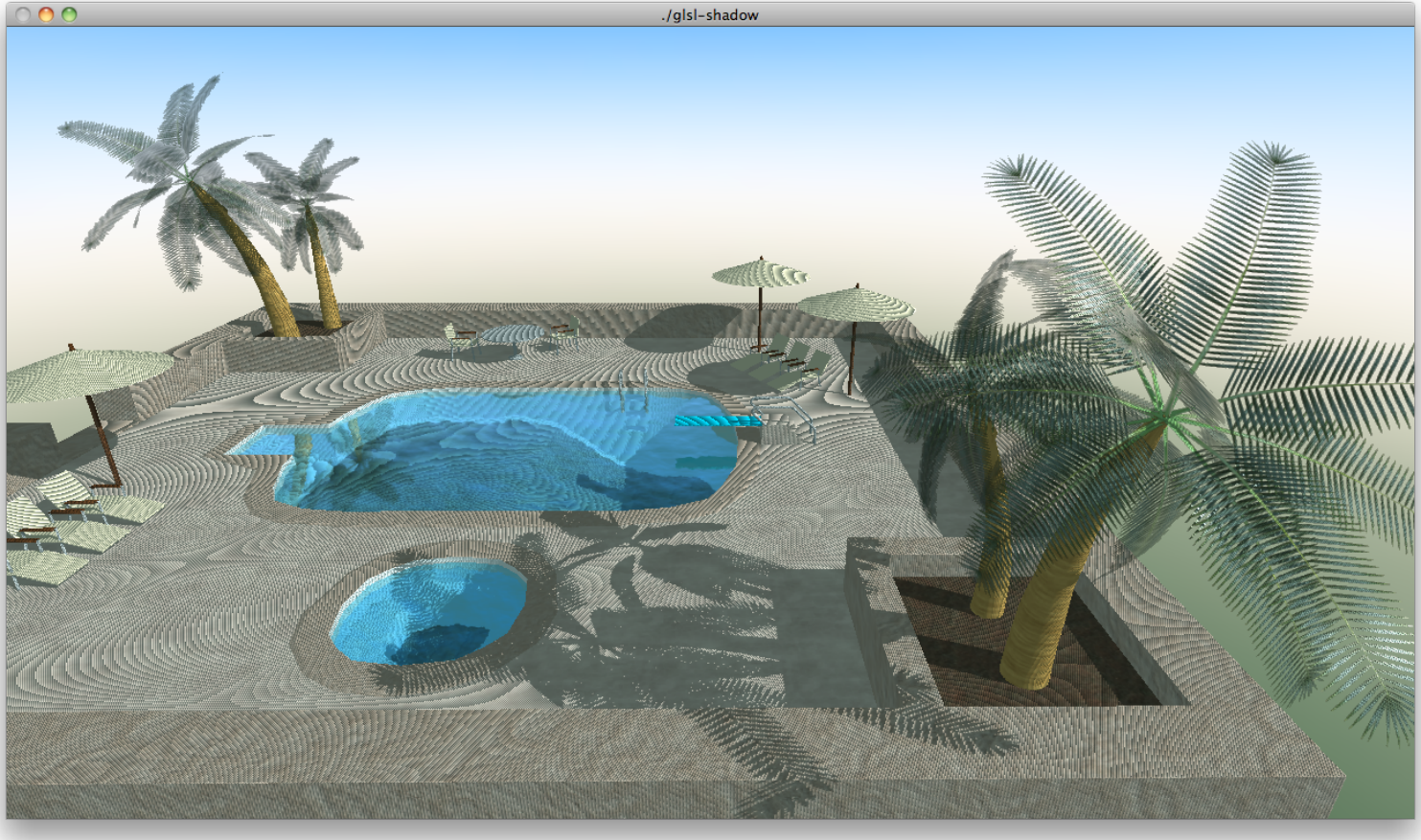
```
uniform sampler2DShadow shadow;

void main()
{
    gl_FragColor = shadow2DProj(shadow, gl_TexCoord[1]);
}
```



The light source depth, in compare-to-texture mode.



The depth-comparison integrated with scene illumination.



You might see moiré artifacts…

Polygon offset

In practice, the precision of the shadow depth buffer does not match that of the shadow map z coordinate.

The solution is to shift all depth buffer values forward very slightly during shadow map rendering.

```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(4.0f, 4.0f);
```

The minimum effective offset value depends upon the shadow map resolution and scene geometry. Choosing it is a black art.

Percentage-closer filtering

Low-resolution shadow maps can be blocky, but we can smooth them in the fragment shader.

Make multiple shadow map references, each offset by a fraction of a texel. Tally the percentage of "hits" and apply the result as a grayscale shadow value.

This is automatic on NVIDIA:

```
glBindTexture(GL_TEXTURE_2D, depth);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```