

TEXTURE MAPPING AND GRAPHICS PIPELINE 4

CS 184: FOUNDATIONS OF COMPUTER GRAPHICS

1 A Texture and Mipmap Adventure

In this problem, we explore texture coordinates and mipmaps. A pixel may cover more or less than one texel, leading to aliasing. To handle this, we use mipmaps—pre-filtered, downscaled versions of the texture.

Consider an image mapped to a texture with a resolution of (128, 128). We'll work through sampling a pixel and retrieving its value from the appropriate mipmap level. Note that the texture coordinates (u, v) lie in the range $[0, 1]$, not the texture resolution range.

1. **Conceptual Question:** What is the resolution of the mipmap at level 3?

Solution: Mipmaps are generated by halving the resolution of the original texture in each dimension. Starting with a texture of size (128, 128), we have:

Level 0: (128, 128)

Level 1: $(128/2, 128/2) = (64, 64)$

Level 2: $(64/2, 64/2) = (32, 32)$

Level 3: $(32/2, 32/2) = (16, 16)$

Thus, the mipmap at level 3 has a resolution of (16, 16).

2. **Conceptual Question:** Derive an expression for the amount of space needed to store all mipmap levels, relative to the memory needed to store the original texture.

Solution: At mipmap level i , the texture resolution is given by $(\frac{x}{2^i}, \frac{x}{2^i})$. Halving each dimension reduces the overall size by a factor of 4. Thus, let x be our texture size of the highest resolution. The total storage required is the sum of the mipmaps, which form a geometric series:

$$x + \frac{x}{4} + \frac{x}{16} + \dots = x \left(1 + \frac{1}{4} + \frac{1}{4^2} + \dots \right)$$

The infinite sum is:

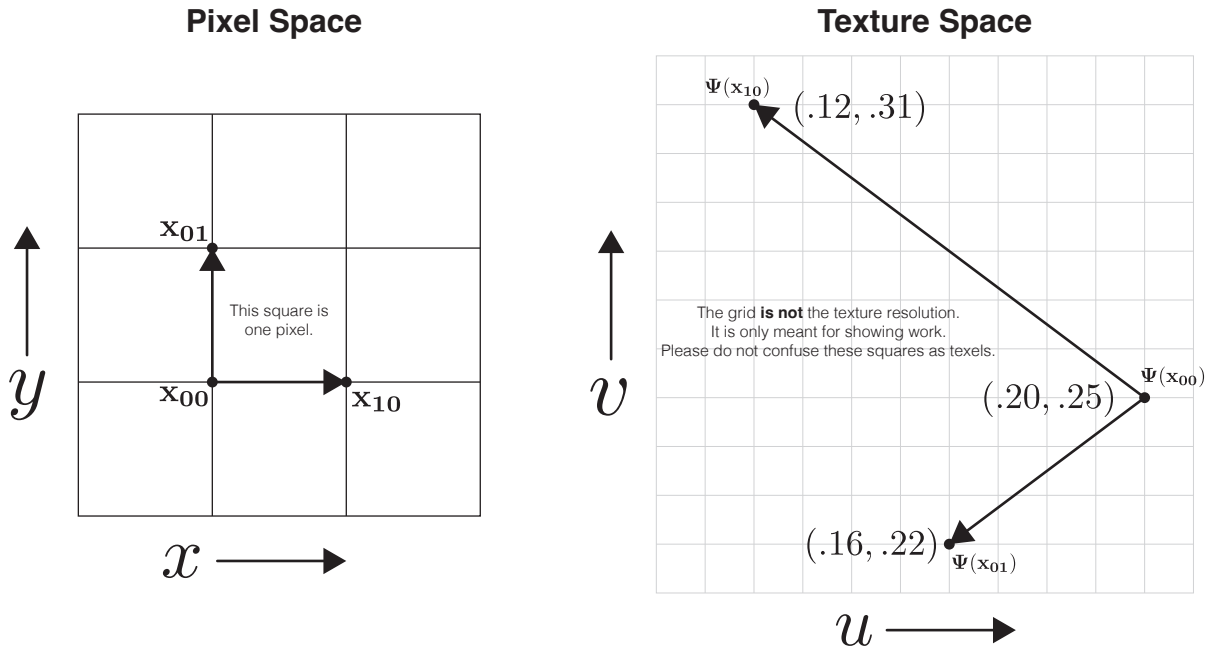
$$\sum_{i=0}^{\infty} \left(\frac{1}{4} \right)^i = \frac{1}{1 - \frac{1}{4}} = \frac{4}{3}.$$

Multiplying by x gives a total of:

$$x \cdot \frac{4}{3} = \frac{4}{3}x.$$

Thus, storing all mipmap levels requires $\frac{4}{3}$ times the memory of the original texture. In practice, the series is finite and stops after the texture resolution is (1×1) , so $\frac{4}{3}x$ is an upper bound.

3. We are given pixel point $\mathbf{x}_{00} = (x, y)$, as well as \mathbf{x}_{01} and \mathbf{x}_{10} , each 1-pixel away from \mathbf{x}_{00} . These three points are mapped to their corresponding locations in texture space by the mapping $\Psi(\mathbf{x})$. Note that the mapping uses barycentric coordinates! The texel space is in the range $[0, 1]$. At what mipmap level, L , should we sample to retrieve the texture for point \mathbf{x}_{00} ?



Solution: We need to compute the length of the vectors in texture space. First, we can compute the derivatives for the unit change in pixel space, $\left(\frac{du}{dx}, \frac{dv}{dx}\right)$ and $\left(\frac{du}{dy}, \frac{dv}{dy}\right)$.

$$\left(\frac{du}{dx}, \frac{dv}{dx}\right) = \Psi(\mathbf{x}_{10}) - \Psi(\mathbf{x}_{00}) = (.12, .31) - (.20, .25) = (-.08, .06) \quad (1)$$

$$\left(\frac{du}{dy}, \frac{dv}{dy}\right) = \Psi(\mathbf{x}_{01}) - \Psi(\mathbf{x}_{00}) = (.16, .22) - (.20, .25) = (-.04, -.03) \quad (2)$$

Next, we find the length of these vectors. Notice that both $(-0.08, -0.06)$ and $(-0.04, -0.03)$ are proportional to $(4, 3)$, which is part of a standard Pythagorean triple $(3, 4, 5)$. Hence, each has magnitude 5 times its scale factor. However, below is a more detailed way to compute it.

$$\left\| \left(\frac{du}{dx}, \frac{dv}{dx}\right) \right\| = \sqrt{(-0.08)^2 + (-0.06)^2} = \sqrt{0.0064 + 0.0036} = \sqrt{0.01} = 0.10, \quad (3)$$

$$\left\| \left(\frac{du}{dy}, \frac{dv}{dy}\right) \right\| = \sqrt{(-0.04)^2 + (-0.03)^2} = \sqrt{0.0016 + 0.0009} = \sqrt{0.0025} = 0.05. \quad (4)$$

Next, let's recall that our texture coordinates are in the range $[0, 1]$ but the texture resolution is $(128, 128)$. We need to scale these derivatives by 128 and then compute the following formula:

$$L = 128 \times \max \left(\left\| \left(\frac{du}{dx}, \frac{dv}{dx} \right) \right\|, \left\| \left(\frac{du}{dy}, \frac{dv}{dy} \right) \right\| \right) \quad (5)$$

$$= 128 \times \max(0.10, 0.05) \quad (6)$$

$$= 128 \times 0.10 \quad (7)$$

$$= 12.8 \quad (8)$$

So, our mipmap selection is

$$D = \log_2(L) = \log_2(12.8) \approx 3.68 \quad (9)$$

4. For mipmap levels 0 through 5, the texture value at $\Psi(x_{00})$ is given by:

$$T_0 = 0.38, \quad T_1 = 0.42, \quad T_2 = 0.36, \quad T_3 = 0.40, \quad T_4 = 0.39, \quad T_5 = 0.37.$$

Given our answer to the previous question, what texture value should we use?

Solution: We found that $D \approx 3.68$ in the previous problem. We round this value to the nearest integer level $\text{round}(3.68) = 4$, and we select from the corresponding texture $T_4 = 0.39$.

5. How can we combine values from two neighboring mipmap levels to get an even smoother result?

Solution: We can use *trilinear interpolation* between the two nearest mipmap levels to $D \approx 3.68$,

$$\lfloor D \rfloor = 3, \quad \lceil D \rceil = 4,$$

First, define the fractional part:

$$\alpha = D - \lfloor D \rfloor = 3.68 - 3 = 0.68.$$

Then we blend between $T_3 = 0.40$ and $T_4 = 0.39$:

$$T_{3.68} = (1 - \alpha)T_3 + \alpha T_4 = 0.32 \times 0.40 + 0.68 \times 0.39 = 0.128 + 0.2652 = 0.3932.$$

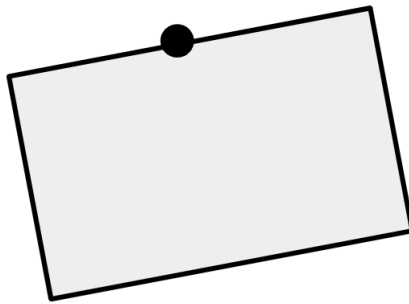
Hence, the final texture value at $D \approx 3.68$ is 0.3932.

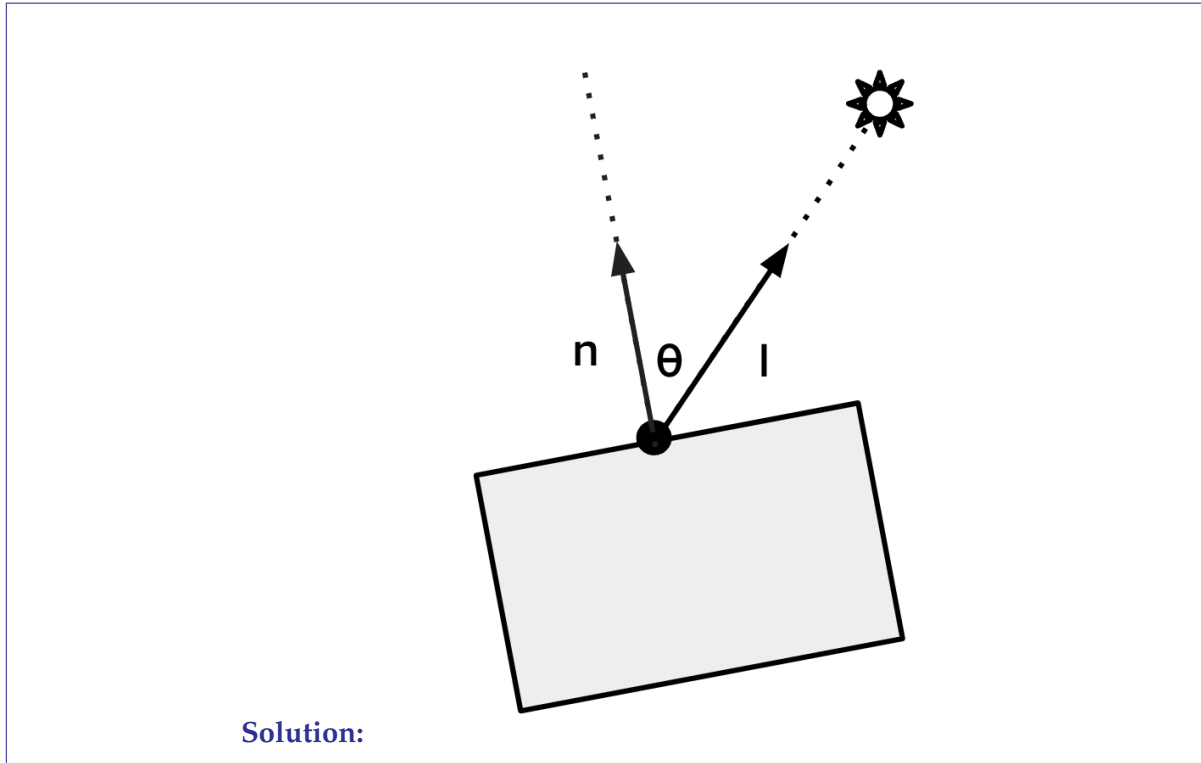
2 Graphics Pipeline — Lightning Round!

1. Name and describe the three terms in the Blinn-Phong Reflection Model.

Solution: Ambient, Diffuse, Specular lighting.

2. A light source shines on a tilted surface. Draw the light direction vector, \mathbf{l} , and the normal vector, \mathbf{n} , at the given point.





3. What is the light per unit area on this surface proportional to, according to Lambert's cosine law?

Solution: $l \cdot n = \cos \theta$. Light intensity reflected off surface is proportional to cosine of the angle between the light direction vector and surface normal vector.

4. Complete the following implementation of the Z-Buffer Algorithm in C++.

```
const int WIDTH = 800; // Width of framebuffer
const int HEIGHT = 600; // Height of framebuffer

struct Color {
    float r, g, b;
};

struct Sample {
    int x, y;
    float z;
    Color color;
};

struct Triangle {
    std::vector<Sample> samples;
};

void zBufferAlgorithm(const _____ triangles,
```

```

        _____ framebuffer,
        _____ zbuffer) {
for (const Triangle& T : triangles) {
    for (const Sample& sample : T.samples) {
        int x = sample.x;
        int y = sample.y;
        float z = sample.z;

        if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
            if (_____ ) {
                framebuffer[x][y] = sample.color;
                _____;
            }
        }
    }
}
}
}

```

Solution:

This solution uses `std::vector`, but other C++ implementations of lists could also work.

- `std::vector<Triangle>&`
- `std::vector<std::vector<Color>>&`
- `std::vector<std::vector<float>>&`
- `z < zbuffer[x][y]`
- `zbuffer[x][y] = z`

5. Prior to running this algorithm, what should the Z-buffer values be initialized to?

Solution:

```
std::numeric_limits<float>::infinity()
```

(Infinity.)