Procedural Textures

CS 385 - Class 22
14 April 2022

Why Procedural?

- Raster textures have limited resolution
  - at some point, you zoom too far, and get blurring
- Procedural textures have infinite resolution
  - of course, if features get too small, you will still get artifacts

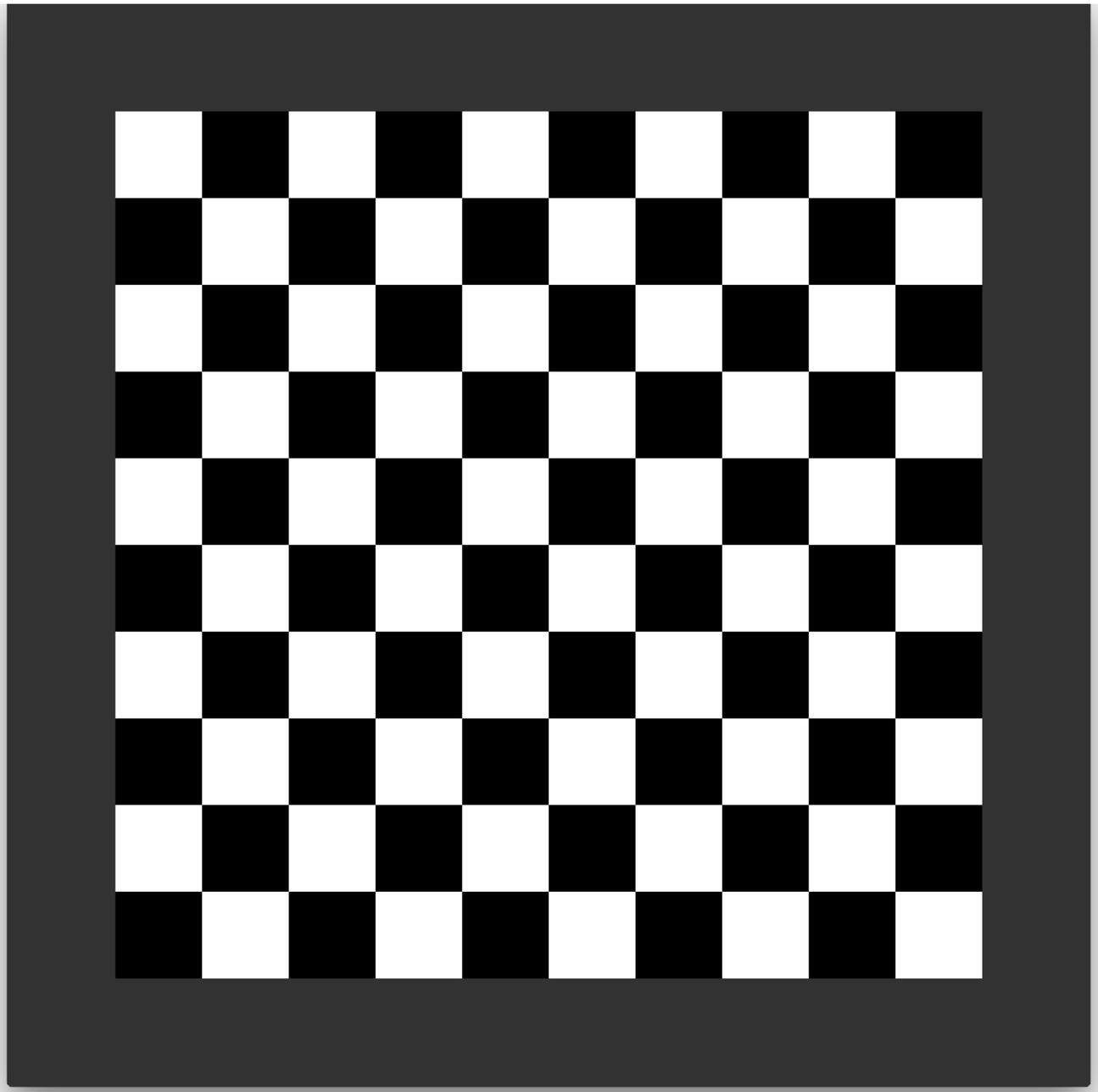
What's a Procedural Texture?

- You've already written several 😝
- It's fancy name for a fragment shader with a specific purpose
  - compute a specific pattern, instead of using a pixel image, for example
- The fragment shader needs to determine the color based on its inputs
  - often we'll use texture coordinates to as parameters into a function that returns the answer
- Use similar tricks you've seen at ShaderToy.com

“Symmetry saves you work”

(Look for repeating patterns)

Checkerboard

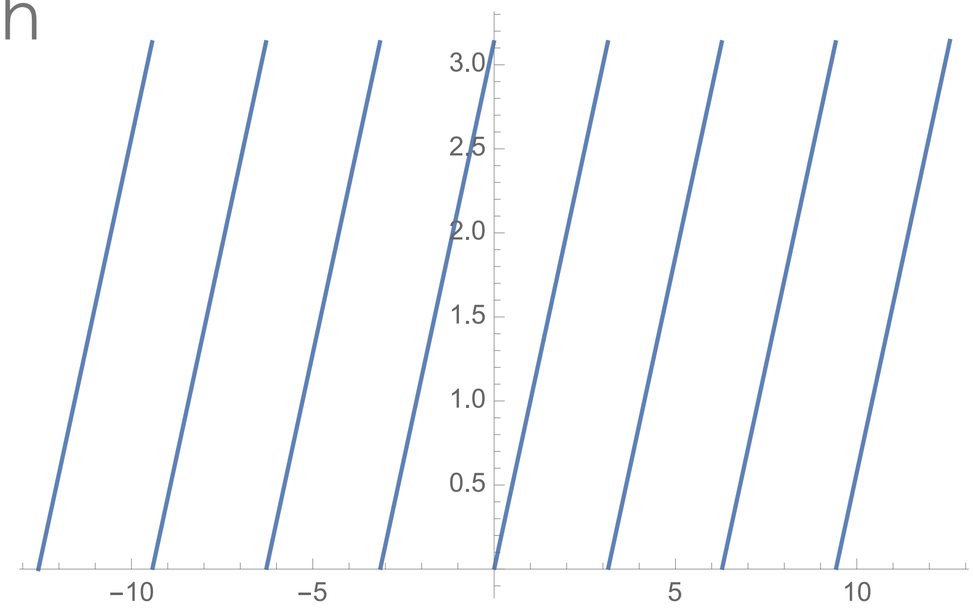


Computing the Checkerboard

- When you see a repeating pattern, think periodic function
  - `mod` (the modulus function) - repeats with a period of the divisor

- or use a sine or cosine function
  - particularly convenient when you can decide on the sign of the value

$$\mathrm{mod}(a, b) = a - \left\lfloor \frac{a}{b} \right\rfloor$$



Graph of `mod(x,π)`

Vertex Shader

- The vertex's position is specified in normalized device coordinates
- Texture coordinates are generated from those vertex positions
  - map vertex coordinate range [-1,1] into texture coordinate range [0, 1]
- Vertex positions are scaled slightly to position the geometry correctly
  - remember that the vertex's x, y, and z coordinates are divided by w
  - reset w to 1.0, otherwise our scale is "removed"

Vertex Shader

```
in  vec4  aPosition;  // in NDCs
out vec2  vTexCoord;

void main()
{
  vTexCoord = 0.5 *
    (aPosition.xy + vec2(1));

  gl_Position = 0.8 * aPosition;
  gl_Position.w = 1.0;
}
```

Fragment Shader

- The fragment shader is run for every fragment in the checkerboard
  - every fragment will get a unique texture coordinate (generated by the rasterizer)
- Shader needs to determine if a particular fragment is inside of a black or white square
  - there are multiple methods for making that determination

Fragment Shader (version 1)

- Use a simple even-odd test (using an integer modulus), and then use an exclusive-or to select the block's color

1. map texture coordinate range into the number of blocks, and then compute even-odd value by doing a `mod` 2 computation
2. select the greyscale color by using an exclusive-or
   - GLSL uses `^^` to represent an exclusive-or for integers

Fragment Shader

```
in  vec2 vTexCoord;
out vec4 fColor;

void main()
{
    const int BlocksPerRow = 10;

  ❶ ivec2 p = ivec2(BlockPerRow * vTexCoord) % 2;

  ❷ fColor.rgb = vec3(p.x ^^ p.y);
    fColor.a   = 1.0;
}
```

Boolean vectors in GLSL

- Boolean vectors contain a truth value per component
  - values are converted during vector construction
- Several vector comparison functions are available
  - `lessThan`, `equal`, `greaterThanEqual`, `notEqual`, etc.
- Decisions can be made based on a vector's components

zero value	FALSE
non-zero value	TRUE

GLSL Function	Result
`any(v)`	return true if any component is true
`all(v)`	return true if all components are true

Fragment Shader (version 2)

- Use a periodic function to define regions and then use an exclusive-or to select the block's color
  - in this case, when both coordinates are these same sign, set the color to white

1. map texture coordinate range into the number of blocks, and then use a sine function to make the region periodic
   - recall that the sine function is periodic over the region 2π; that creates two blocks per period, so we need to take that into consideration

$$\text{rowSize} = \frac{2\pi\ \text{BlocksPerRow}}{2}$$

Fragment Shader

```
in  vec2 vTexCoord;
out vec4 fColor;

void main()
{
    const int BlocksPerRow = 10;
    const float Pi = 3.141592654;
    const float rowSize = BlocksPerRow * Pi;

  ❶ vec2 p = sin(rowSize * vTexCoord);

  ❷ bvec2 b = bvec2( step(0.0, p) );

    fColor.rgb = vec3( b.x ^^ b.y );
    fColor.a = 1.0;
}
```

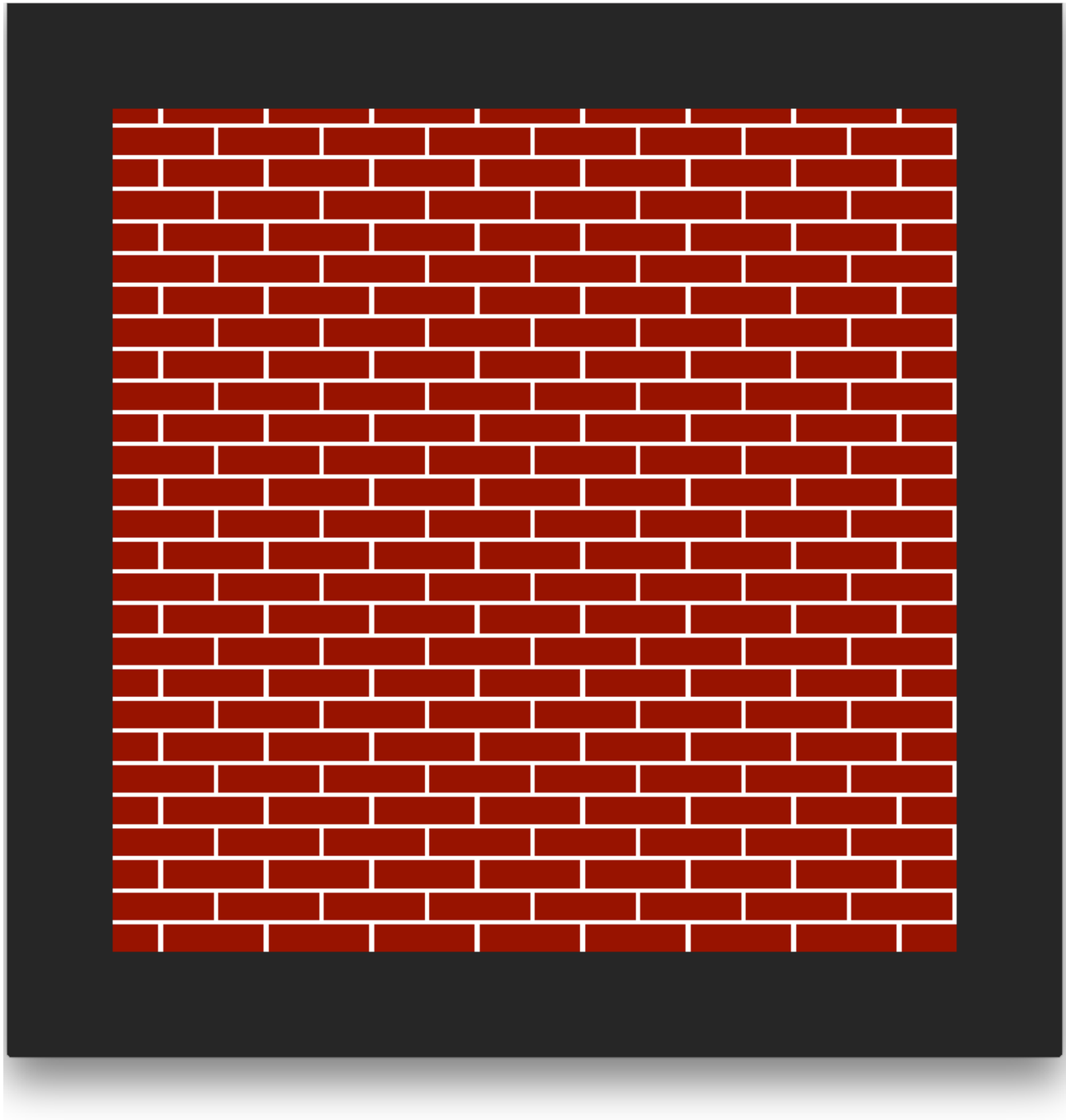
Fragment Shader (version 2)

2. create a boolean vector where the value indicates if the coordinate is negative.
   - use the `step` function to determine the sign of each component of *p*
   - another function `lessThan`, would do the same thing
     - the only difference between the two is that `lessThan` requires a vector for comparison, while `step` supports providing a scalar value that's used for each component of p

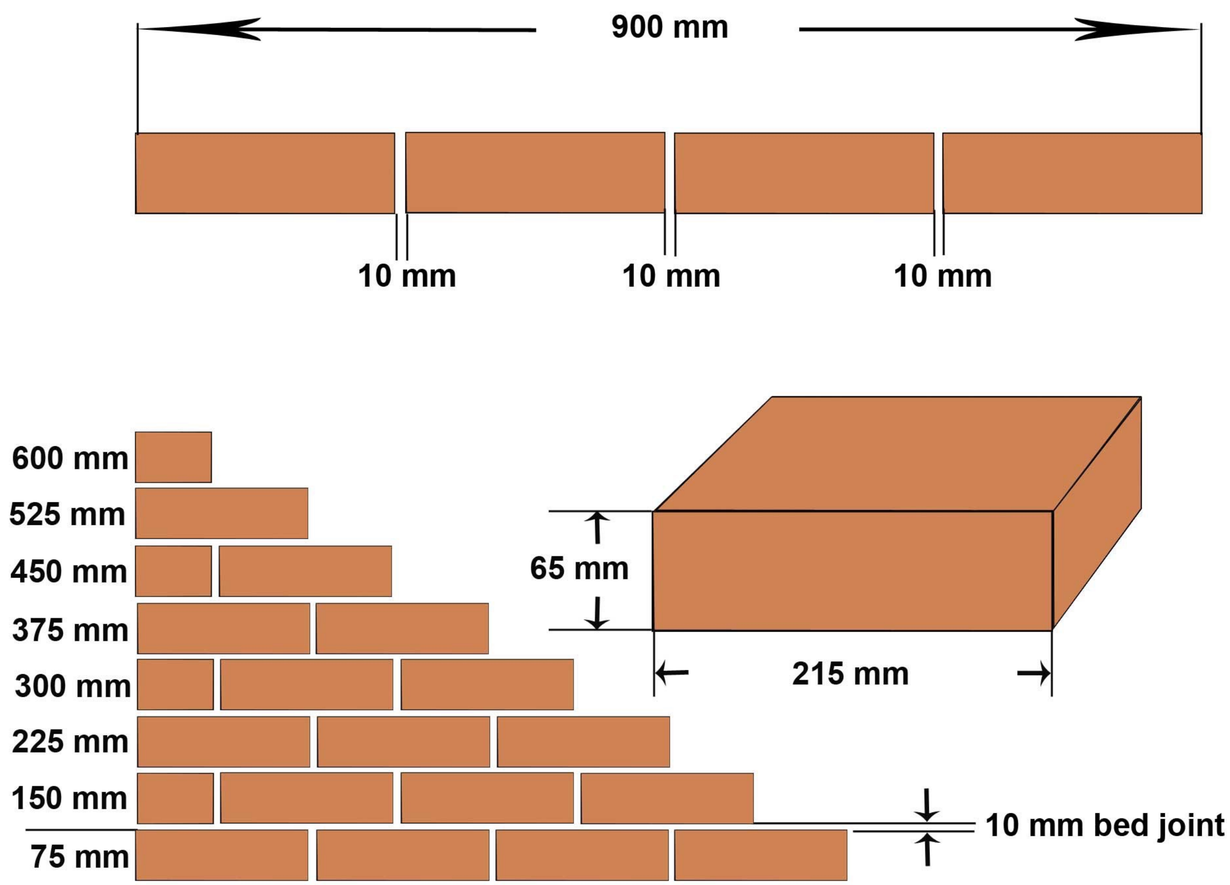
Fragment Shader

```
in  vec2 vTexCoord;
out vec4 fColor;

void main()
{
    const int BlocksPerRow = 10;
    const float Pi = 3.141592654;
    const float rowSize = BlocksPerRow * Pi;

  ❶ vec2 p = sin(rowSize * vTexCoord);

  ❷ bvec2 b = bvec2( step(0.0, p) );

    fColor.rgb = vec3( b.x ^^ b.y );
    fColor.a = 1.0;
}
```

Bricks



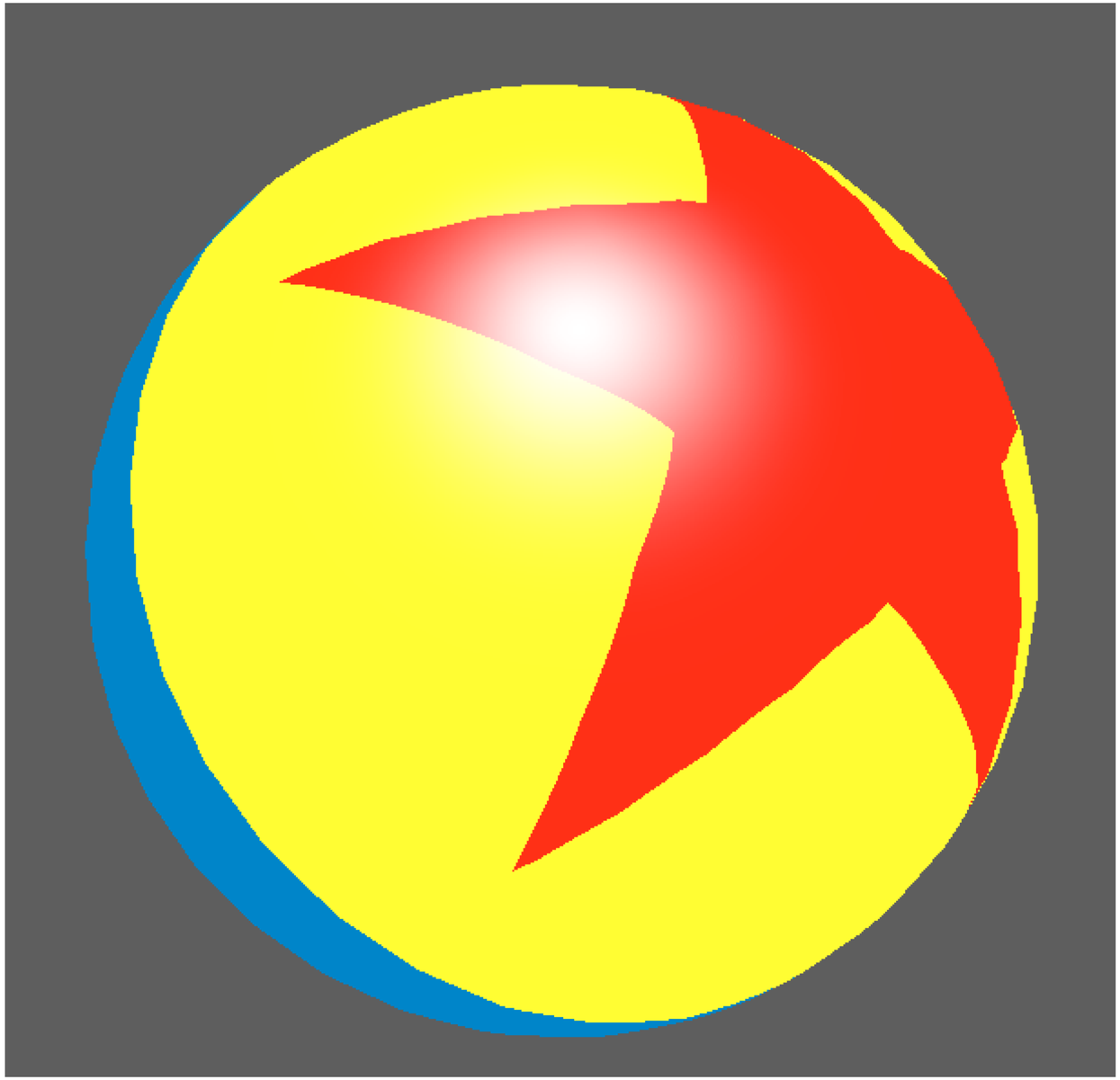
How big is a brick?



A Little Motivation

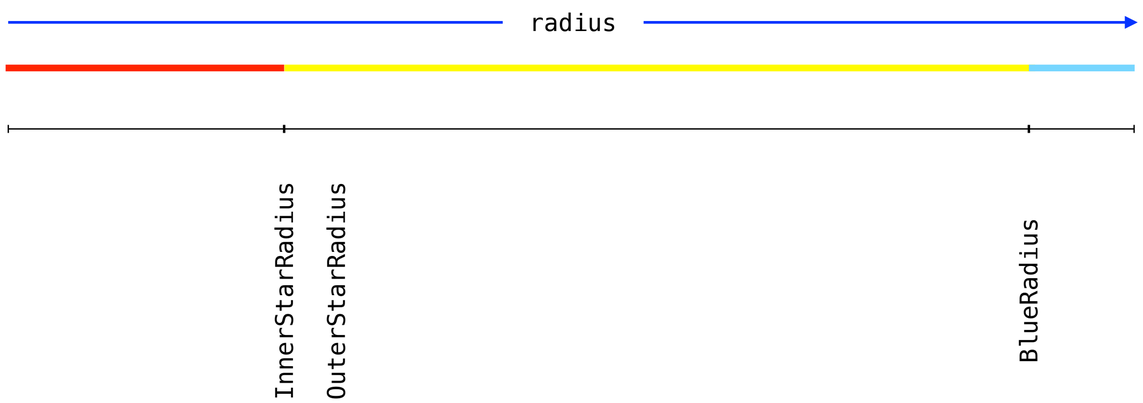
Texturing Luxo's Ball

- Assume the ball is symmetric around the z axis
  - i.e., there's a star at each pole of the sphere
  - the blue stripe goes around the equator
- Use the distance from the pole to the equator as the radius



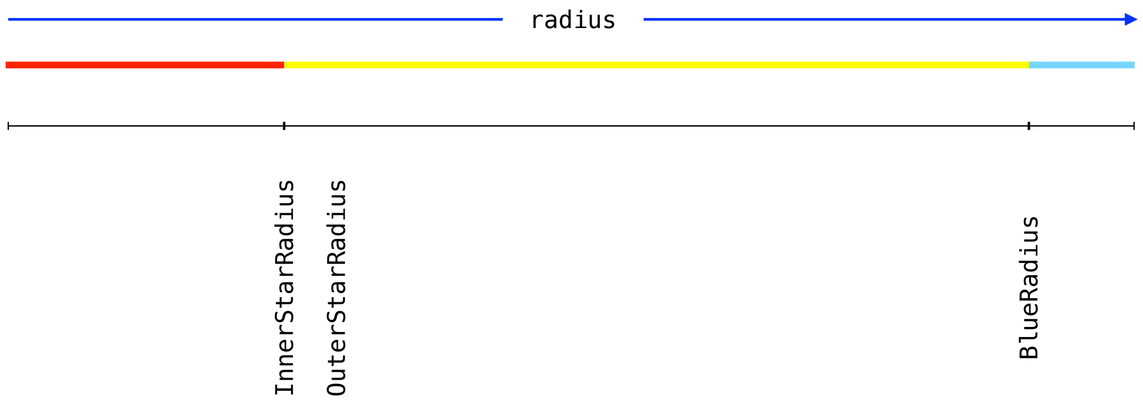
Radial Symmetry

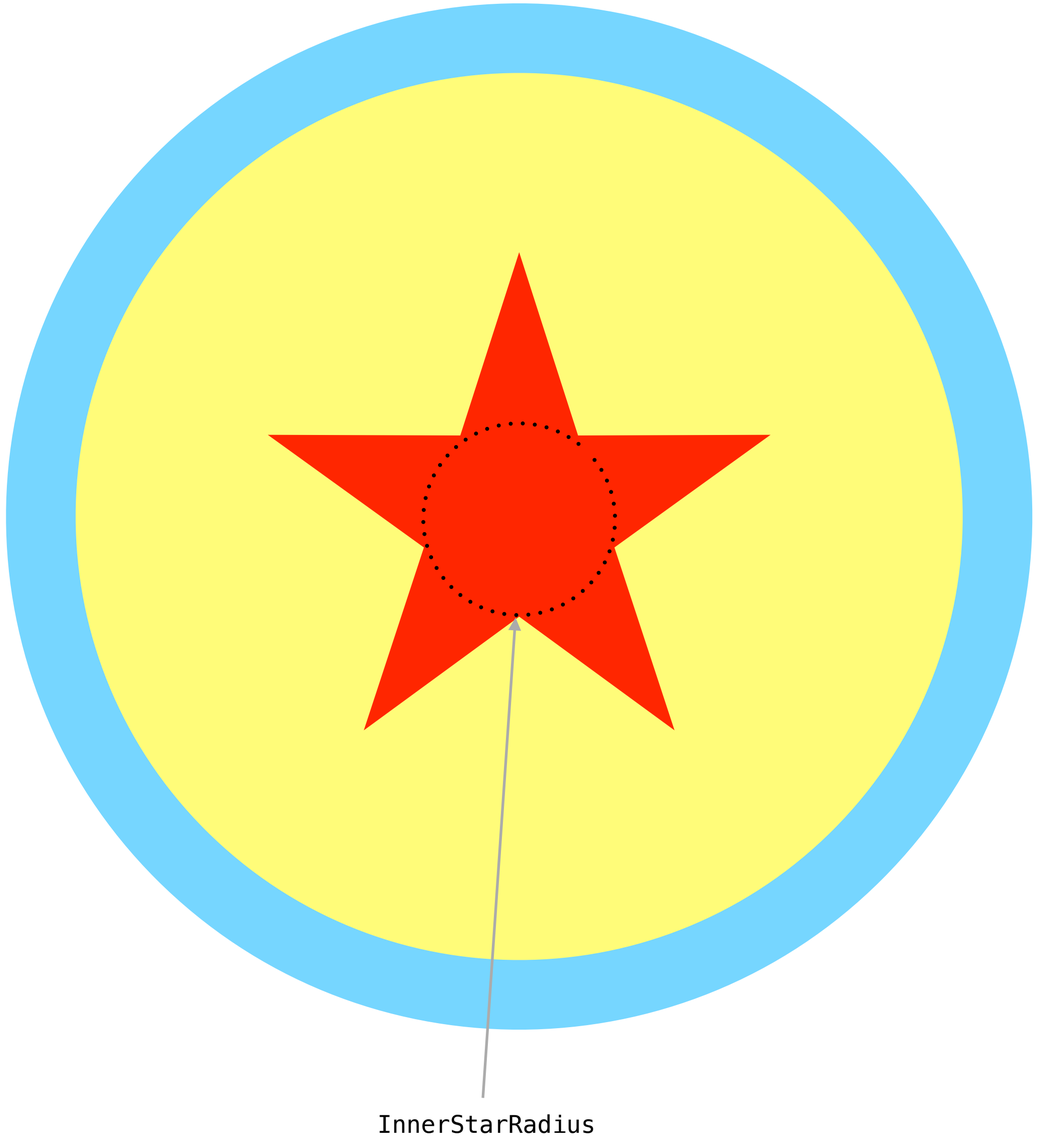
- The ball’s texture is radially symmetric
  - the colors are the same* regardless of the angle around the center
  - use the radius to choose which color to select



Radial Symmetry

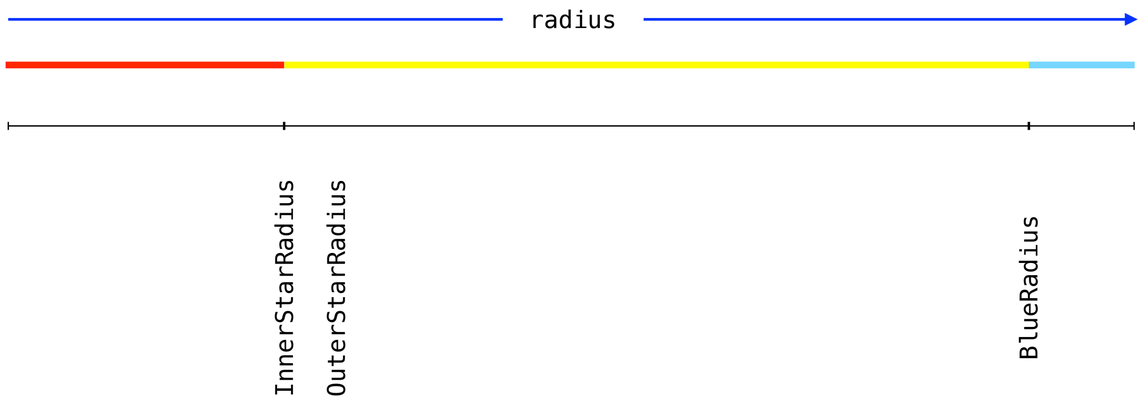
- The ball's texture is radially symmetric
  - the colors are the same* regardless of the angle around the center
  - use the radius to choose which color to select

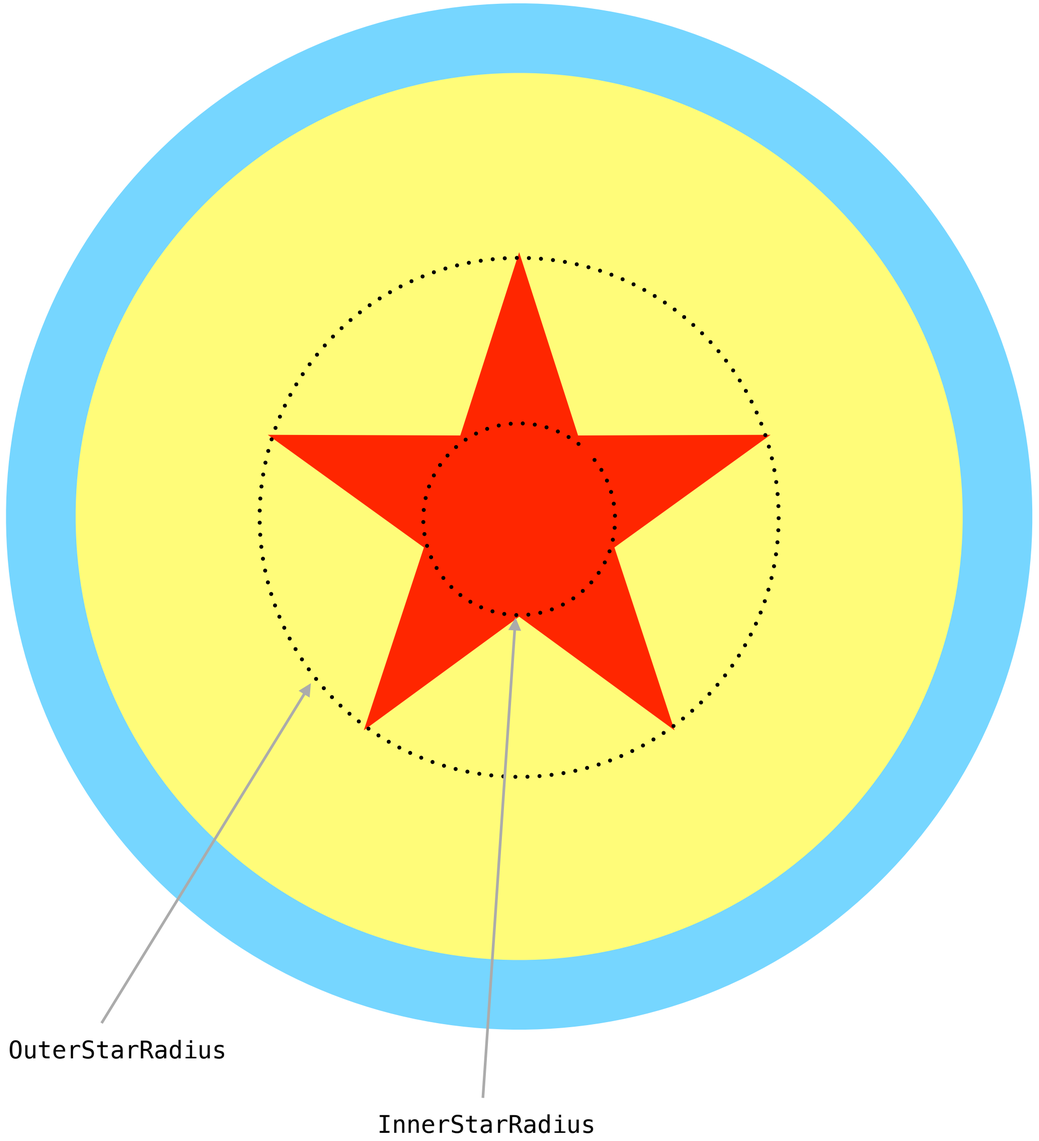




Radial Symmetry

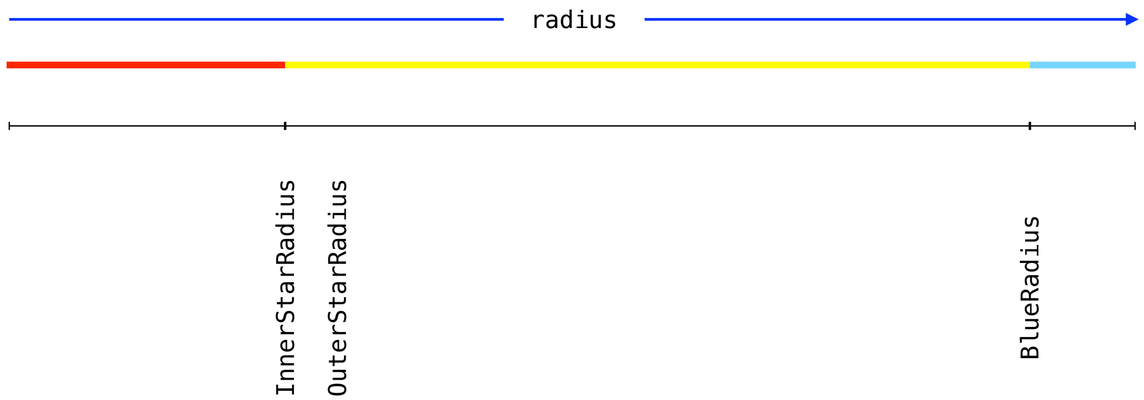
- The ball's texture is radially symmetric
  - the colors are the same* regardless of the angle around the center
  - use the radius to choose which color to select

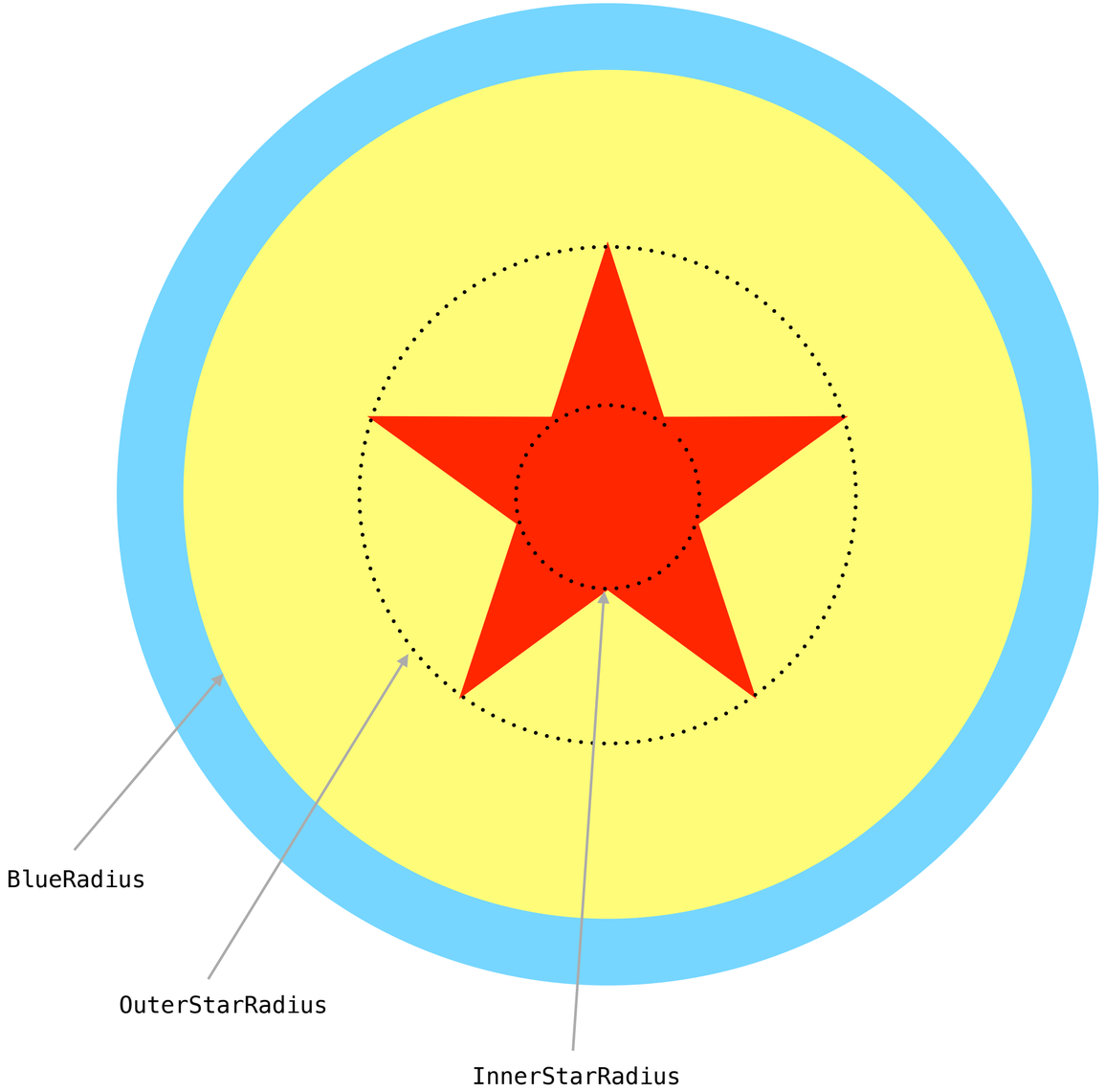




Radial Symmetry

- The ball's texture is radially symmetric
  - the colors are the same* regardless of the angle around the center
  - use the radius to choose which color to select



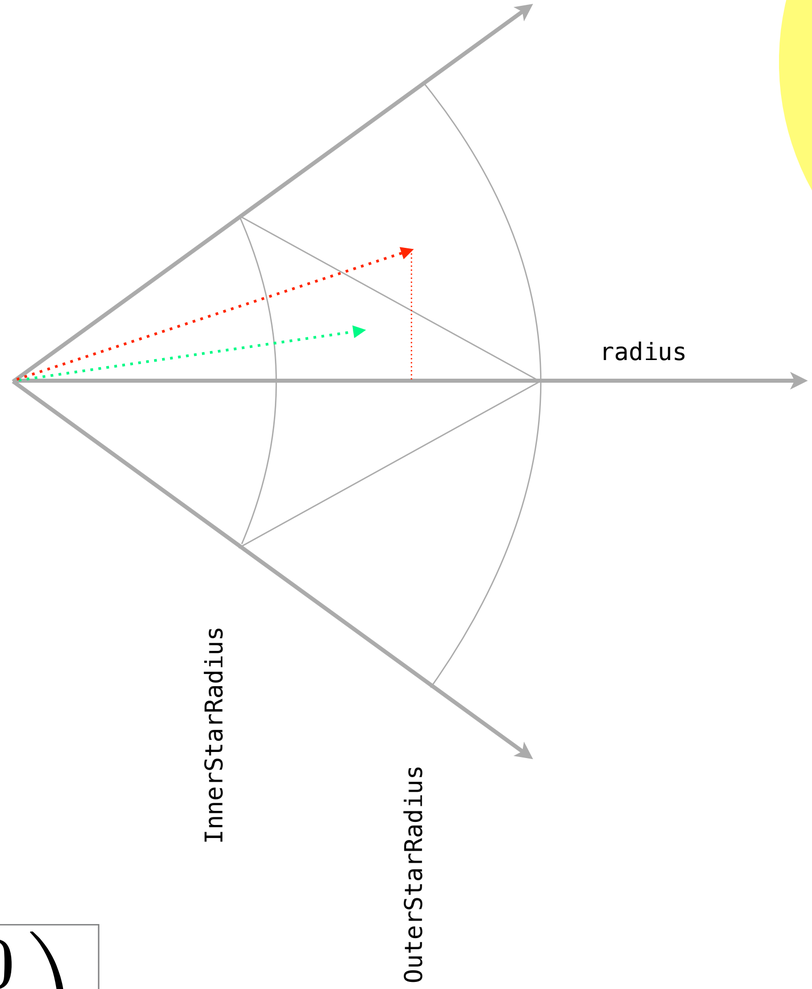


Star Symmetry

- The star has five-fold radial symmetry
  - i.e., the texture can be cut into five equally-sized slices radially (think a slice of pizza)
- the pattern is angularly periodic — it repeats every 72° (360° / 5)

$$\text{angle} = \text{mod}(\text{angle}, 72.0)$$

  - further, each slice is symmetric around its center line
  - subtracting half of the period angle and taking the absolute value simplifies the problem



$$\text{angle} = \text{abs}\left(\text{mod}(\text{angle}, 72.0) - \frac{72.0}{2}\right)$$

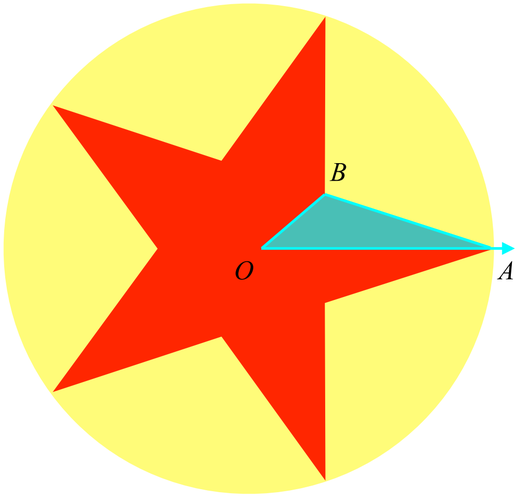
Drawing the Star

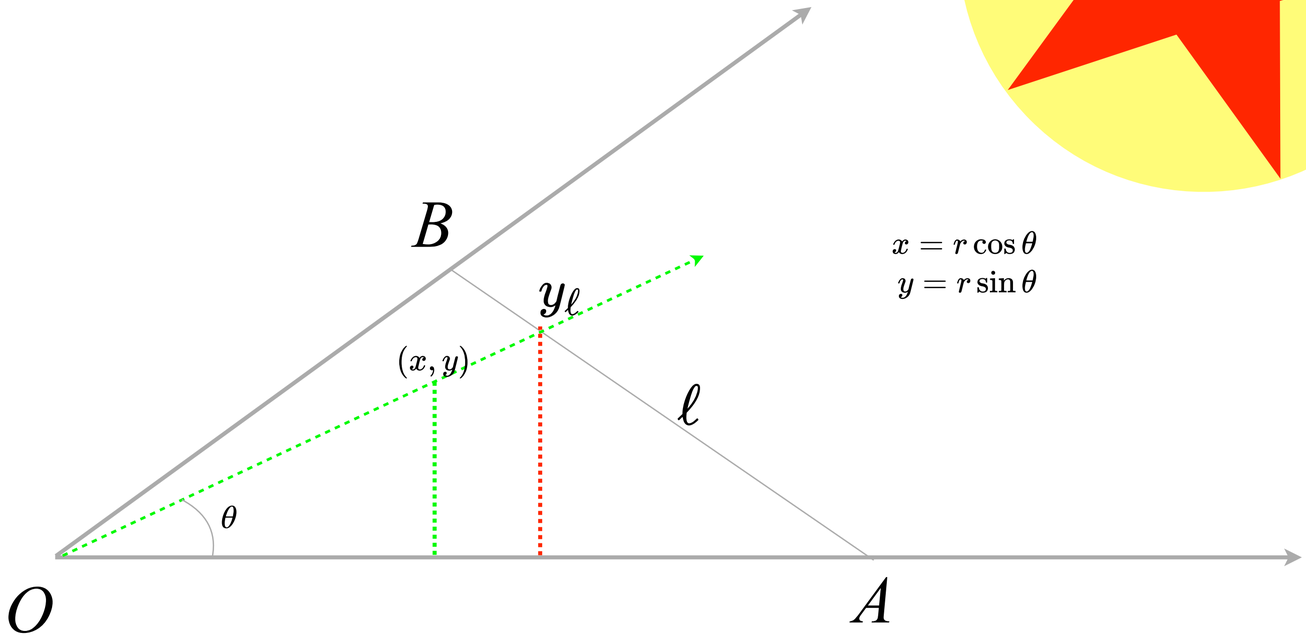
- Need to determine if the point is inside the star
  - this reduces to determining if the point's y coordinate is less than the same point on the star's edge (described by the line ℓ)
  - ℓ is the line between A and B
  - first, use the point-point formula for a line

$$y - A_y = \underbrace{\frac{B_y - A_y}{B_x - A_x}}_{m}(x - A_x)$$

  - then, define the line in terms of the slope-intercept form of the line

$$y = mx + b \qquad b = -m\,A_x + A_y$$





$$x = r\cos\theta$$
$$y = r\sin\theta$$

$$A = (\text{OuterStarRadius}, 0.0)$$
$$B = \left(\text{InnerStarRadius}\left(\cos\frac{72}{2}, \sin\frac{72}{2}\right)\right)$$

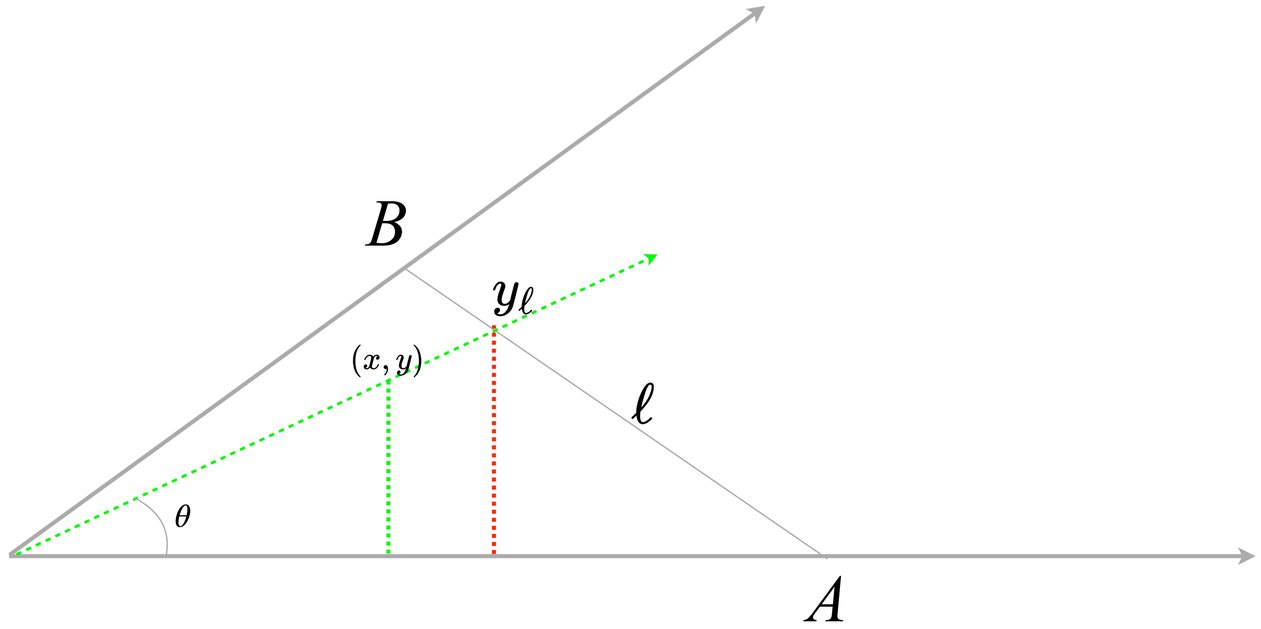
Drawing the Star (cont'd)

- With the equation of the line ℓ, compute the (x, y) relative to the $\mathbf{angle}$

$$\mathbf{angle} = \mathbf{abs}\left(\mathbf{mod}(\mathbf{angle}, 72.0) - \frac{72.0}{2}\right)$$

$$x = r\cos\theta$$
$$y = r\sin\theta$$

- then evaluate the line equation at the appropriate x coordinate, and compare y values to determine the color



Fragment Shader

- Specify a number of constants for shading the ball

1. Compute the distance from the center of the star
   - this value is used to determine which color we'll use for the radially symmetric part
2. Compute the angle of the original texture coordinate
   - this value will be reduced into the periodic domain in step 3.

Fragment Shader

```
in  vec2 vTexCoord;
out vec4 fColor;

void main()
{
    float x = vTexCoord.x;
    float y = vTexCoord.y;

  ❶ float radius = length( vTexCoord );
  ❷ float angle = degrees( atan( y, x ) );

    const vec4 blue = vec4( 0, 0, 1, 1 );
    const vec4 red = vec4( 1, 0, 0, 1 );
    const vec4 yellow = vec4( 1, 1, 0, 1 );

    const float blueRadius = 0.8;
    const float OuterStarRadius = 0.6;
    const float InnerStarRadius = 0.2;
```

Fragment Shader

3. Map the angle into the periodic domain
4. Compute the points used in the star's edge line equation ℓ
5. Compute the y value on the star's edge for the point's x coordinate
6. Compare the point's y coordinate with the star's edge, and choose the appropriate color

Fragment Shader

```
    if ( radius > blueRadius ) {
        fColor = blue;
    }
    else if ( radius > OuterStarRadius ) {
        fColor = yellow;
    }
    else if ( radius < InnerStarRadius ) {
        fColor = red;
    }
    else {
      ❸ angle = abs( radians( mod(angle, 72) - 36 ) );

      ❹ vec2 A = vec2( OuterStarRadius, 0 );
        float theta = radians(36);
      ❹ vec2 B = InnerStarRadius *
            vec2(cos(theta), sin(theta));

        float m = (B.y - A.y)/(B.x - A.x);
        float b = -m * A.x;

        x = radius * cos(angle);

     ❺ float lineY = m * x + b;

        y = radius * sin(angle);

     ❻ fColor = y < lineY ? red : yellow;
    }
}
```