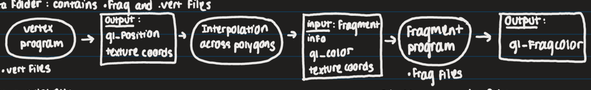
Project 4 - Full Guide

Tuesday, November 15, 2022 6:37 PM

Overview

- data folder : contains .frag and .vert files



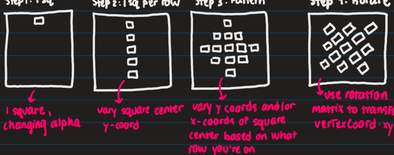
.vert files .frag files

In a nutshell:

- Altering geometry (normals)? → .vert file
  - ↳ Ripples
- Altering color? → .frag file — Per "vertexCoord" basis
  - ↳ Squares, blur, and fractal

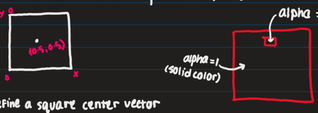
Ripples → .pde file
↳ Subdivide the quad

Squares



Step 1: Create 1 square:

↳ Note that the center of the grid is (0.5, 0.5)



1. Define a square center vector
2. Check to see if square center.x is within a certain distance of vertexCoord.x
   AND if square center.y is within a certain distance of vertexCoord.y
   ↳ Distance: Dimension of your square side
   MUST check the two components SEPARATELY (otherwise, you'll end up with a circle!)
3. If this is the case, set alpha to 0 (making the color transparent at that point)
   ↳ alpha set to 1 be default

gl_FragColor(r, g, b, alpha)

Step 2: create a column of squares

↳ Hint: Use a loop and change the square center's y-coordinate with each iteration

- You're performing the same check as Step 1, just multiple times / in the for-loop

Step 3: Upright pattern

↳ conditionally draw a certain # of squares based on what row you're on
↳ Hint: Use a second loop (think of the board) and change square center's x-coord accordingly

Step 4: Rotate pattern

1. Define a rotation matrix (mat2)
2. vec2 tempCoord → vertexCoord translated with respect to the center of the grid
3. tempCoord → rotation * tempCoord
4. tempCoord → tempCoord translated back to center of the grid
5. Replace your vertexCoord checks (i.e. the condition to set alpha to 0) with tempCoord
   ↳ checking for squares in the "transformed" positions

Blur Mask

use a grid to sample the color of the neighboring texels
↳ start with a 3x3 grid, then vary the grid size based on the mask intensity

Pseudocode (to start):

```
blur_col : (0,0,0,0)        → Running "average of blur color"
blur_radius : 3
texel_size : 1 / size of texture   ↗ Hint: look into "textureSize" function in GLSL
  For i: -blur_radius, blur_radius, inc by 1
    For j: -blur_radius, blur_radius, inc by 1
      sample texture at: (vertexCoord.x + i * texel_size,   → texture2D call (texture2D grabs r,g,b components of that sampled texel)
                          vertexCoord.y + j * texel_size)
  blur_color /= (blur_radius * blur_radius)   → average color across grid
  set diffuse_color to blur_color
```

Calculating Intensity of Mask

↳ take any of mask_color's r,g,b components

- ≤ 0.1 → lots of blur
- 0.1 < x ≤ 0.5 → Medium blur
- 0.5 > 7 → no blur

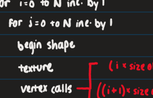
Set blur radius value based on mask intensity

*It's OKAY if you have a "crack" in the image. We will not take points off for that

Ripples

1. Subdividing the quad into a N x N grid (40 ≤ N ≤ 60)
   ↳ p4_gpu.pde
2. Displace the vertices along the normal vector using a sinusoidal pattern
   ↳ bumps.vert

Step 1: Subdividing the quad



Pseudocode (under the bumps_shader in the .pde)

```
push matrix
translate                 ← provided to you
size of quad = S
For i=0 to N inc. by 1
  For j=0 to N inc. by 1
    begin shape
      texture
      vertex calls   (i x size of quad), (j x size of quad), 0, (i/N), (j/N)
                     ((i+1) x size of quad), j x size of quad, 0, (i+1)/N, (j/N)
                     ((i+1) x size of quad), (j+1) x size of quad, 0, (i+1)/N, (j+1)/N
                     (i x size of quad), (j+1) x size of quad, 0, (i/N), (j+1)/N
      end shape
    endFor
  endFor
pop matrix
```

Step 2: Pushing the normal vectors out

↳ Since we're editing the geometry of the quad, we edit the bumps.vert file

1. Find distance between vertexCoord and center of the quad (0.5, 0.5)
2. Set offset to (sin(distance * factor) + 1) / 2
   ↳ factor can be any value depending on how much you want to push out the normal (e.g. 50)
3. Create a shift vector: (Factor * offset) * normal
   (scalar) (vector)
4. Add the shift vector to vertex and apply transform
   ↳ transform * (vertex + shift vector)

Fractal

- Process:
  1. calculate complex numbers c and z
  2. update calculation of z over 20 iterations
  3. check to see if z's length is within the radius of 50
     a. Yes → color white
     b. No → keep the color as red

Step 1: Calculate complex numbers: c and z

keep within range of [-3.14, 3.14]

c = < cx, cy >

z = < vertexCoord.x * 6.28 - 3.14, vertexCoord.y * 6.28 - 3.14 >

Step 2: Update calculation of z for 20 iterations

For i=0 to 20:

sin(z.real) cosh(z.imag) cos(z.real) sinh(z.imag)

complex sine vector = < sin(z.x) * cosh(z.y), cos(z.x) * sinh(z.y) >

z = < c.x * complex sine vector.x − c.y * complex sine vector.y,   ← x-component
      c.x * complex sine vector.y + c.y * complex sine vector.x >   ← y-component

↳ this is the "z_1 = c * sin(z_0)" calculation, per iteration

Step 3: Radius check with z

diffuse color set to red
if z's length is < 50^2   ← radius
  set diffuse color to white   ← masking the pattern here