VGP352 – Week 9

- Agenda:
  - Quiz #4
  - Final in-class presentation
  - Procedural textures
    - Animated height maps
    - Generating normal maps from height maps

Animating Height-map – Water

- We want to animate a height-map that represents small waves

Animating Height-map – Water

- We want to animate a height-map that represents small waves
  - Simulate this as a mesh of particles connected by springs
  - Each water particle is “pulled” up or down by surrounding water

Animating Height-map – Water

- We want to animate a height-map that represents small waves
  - Simulate this as a mesh of particles connected by springs
  - Each water particle is “pulled” up or down by surrounding water
- Track various data for simulation:
  - Store wave height in R of texture
  - Store wave velocity in G of texture
  - Wave “mass”, spring constants, and time step are uniforms

Animating Height-map – Water

- Springs apply a force, f_s, proportional to their extension
  - Force applied to a water element by one of its neighbors is:

$$f_s = \Delta h \times K_s$$

Difference in height → Δh; Spring constant → K_s

Animating Height-map – Water

- Springs apply a force, f_s, proportional to their extension
  - Force applied to a water element by one of its neighbors is:

$$f_s = \Delta h \times K_s$$

Difference in height

Spring constant

  - Updated velocity is:

$$V_n = \frac{\Delta t \times \sum f_s}{m} + V_{n-1}$$

Elapsed time

Mass of water

Animating Height-map – Water

- Springs apply a force, f_s, proportional to their extension
  - Updated position is:

$$H_n = \Delta t \times V_n + H_{n-1}$$

Animating Height-map – Water

- With no other forces, this simulation would oscillate forever

Animating Height-map – Water

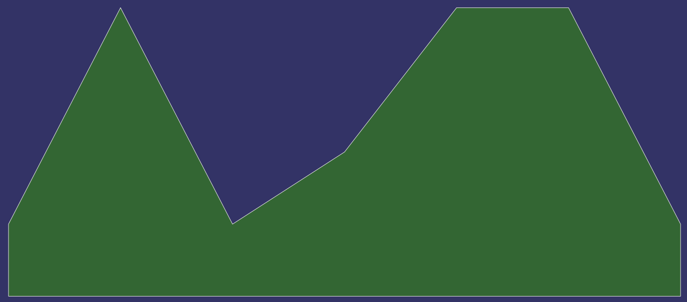
- With no other forces, this simulation would oscillate forever
  - Add one more “virtual” spring to pull each water particle to 0.5
  - This spring should have a *very* small constant

Animating Height-map – Water

```
void main(void)
{
    vec4 me = texture2D(wave_state, gl_TexCoord[0].xy);
    vec2 f_vec = vec2(-4.0 * me.x, 0.5 - me.x);

    f_vec.x += texture2D(wave_state, north).r;
    f_vec.x += texture2D(wave_state, south).r;
    f_vec.x += texture2D(wave_state, east).r;
    f_vec.x += texture2D(wave_state, west).r;

    float F = dot(spring_constant, f_vec);
    float V = (time_over_mass * F) + (me.y - 0.5);
    float H = (time * V) + me.x;

    gl_FragColor = vec4(H, V + 0.5, 0.0, 0.0);
}
```

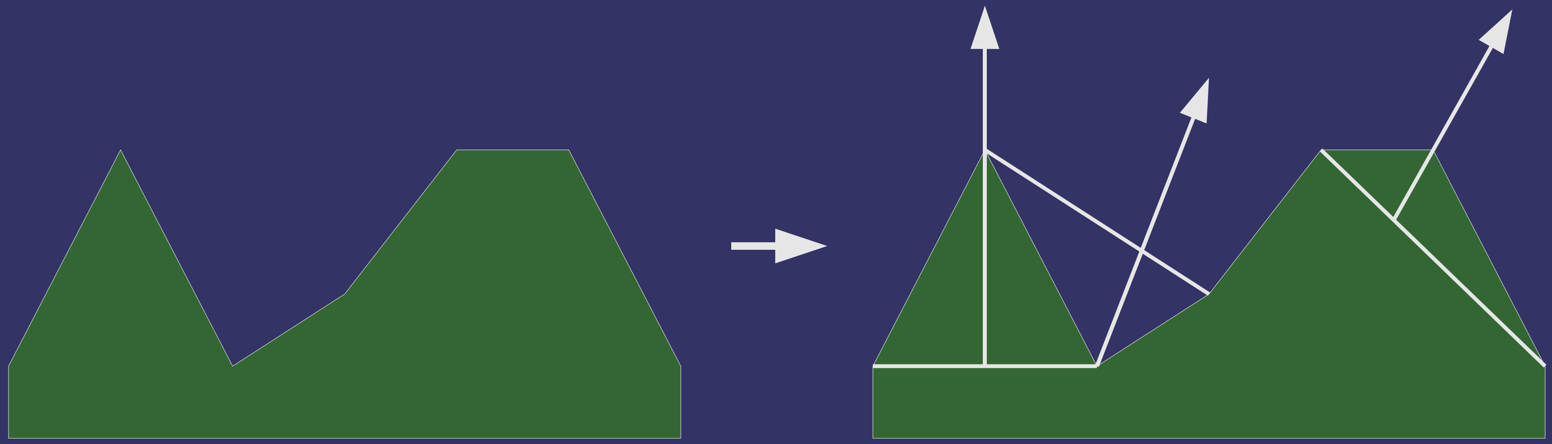
Convert Height-map to Normal-map

- Given a height-map (true bump-map), generate the corresponding normal-map



Convert Height-map to Normal-map

- Given a height-map (true bump-map), generate the corresponding normal-map
  - The X component of the normal is the inverse slope of the line between the east and west neighbors
  - Likewise for the Y component and the north and south neighbors



Convert Height-map to Normal-map

- Task ideally suited to fragment shader!

Convert Height-map to Normal-map

- Task ideally suited to fragment shader!
  - Using render-to-texture, draw a single, texture-sized quad with texture coordinates ranging from (0, 0) to (1, 1)

Convert Height-map to Normal-map

- Task ideally suited to fragment shader!
  - Using render-to-texture, draw a single, texture-sized quad with texture coordinates ranging from (0, 0) to (1, 1)
  - At each fragment read the 4 neighbor texels
    - Call them n, s, e, and w
    - Be careful of texture coordinate wrap modes
    - Apply scale factor to exaggerate bumpiness

Convert Height-map to Normal-map

- Task ideally suited to fragment shader!
  - Using render-to-texture, draw a single, texture-sized quad with texture coordinates ranging from (0, 0) to (1, 1)
  - At each fragment read the 4 neighbor texels
    - Call them *n*, *s*, *e*, and *w*
    - Be careful of texture coordinate wrap modes
    - Apply scale factor to exaggerate bumpiness
  - Normal direction is:

```
vec3 a = vec3(0.0, scale, w.x - e.x);
vec3 b = vec3(scale, 0.0, n.x - s.x);
vec3 n = normalize(cross(b, a));
```

Convert Height-map to Normal-map

- Task ideally suited to fragment shader!
  - Using render-to-texture, draw a single, texture-sized quad with texture coordinates ranging from (0, 0) to (1, 1)
  - At each fragment read the 4 neighbor texels
    - Call them n, s, e, and w
    - Be careful of texture coordinate wrap modes
    - Apply scale factor to exaggerate bumpiness
  - Normal direction is:

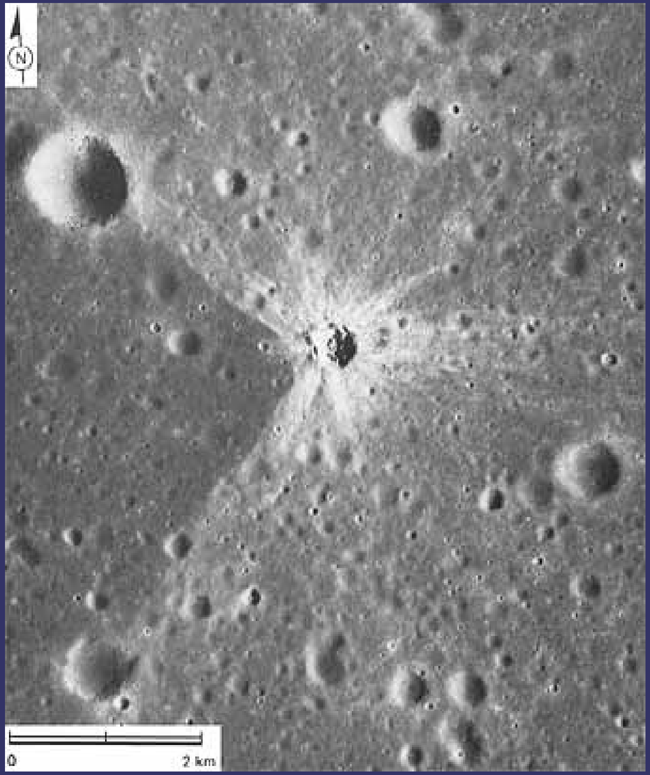
```
vec3 a = vec3(0.0, scale, w.x - e.x);
vec3 b = vec3(scale, 0.0, n.x - s.x);
vec3 n = normalize(cross(b, a));
```

  - Convert components to [0, 1] range and write to `gl_FragColor`

Break

Crater Shader

- Task: create a procedural texture for impact craters on, for example, the moon

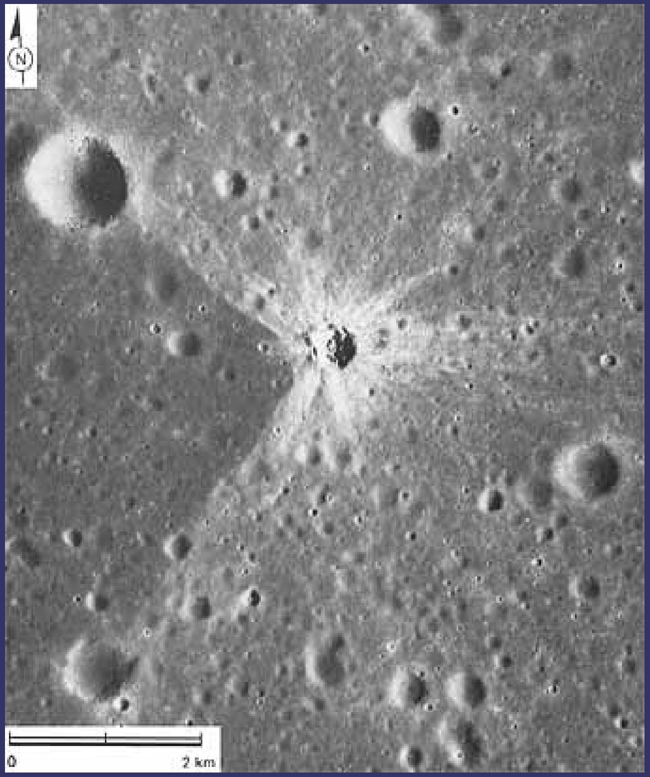


Original image from http://www.hq.nasa.gov/office/pao/History/SP-362/ch5.2.htm

4-March-2008

Crater Shader

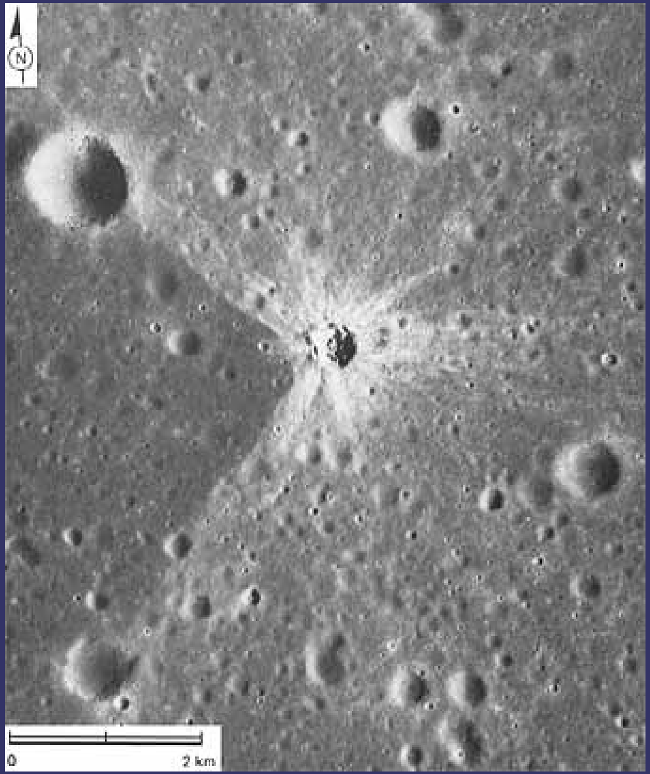
- Two parts to this shader



4-March-2008

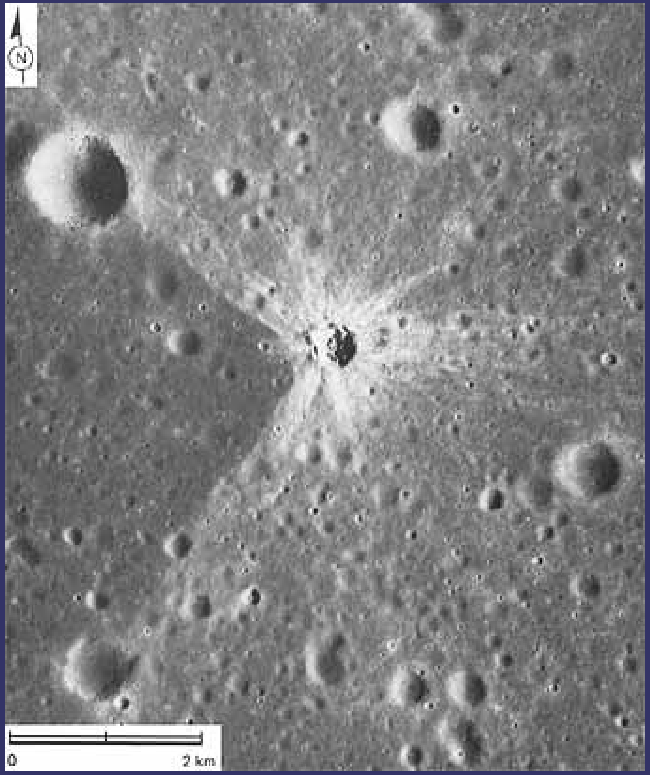
Crater Shader

- Two parts to this shader
  - Height / normal
  - Color



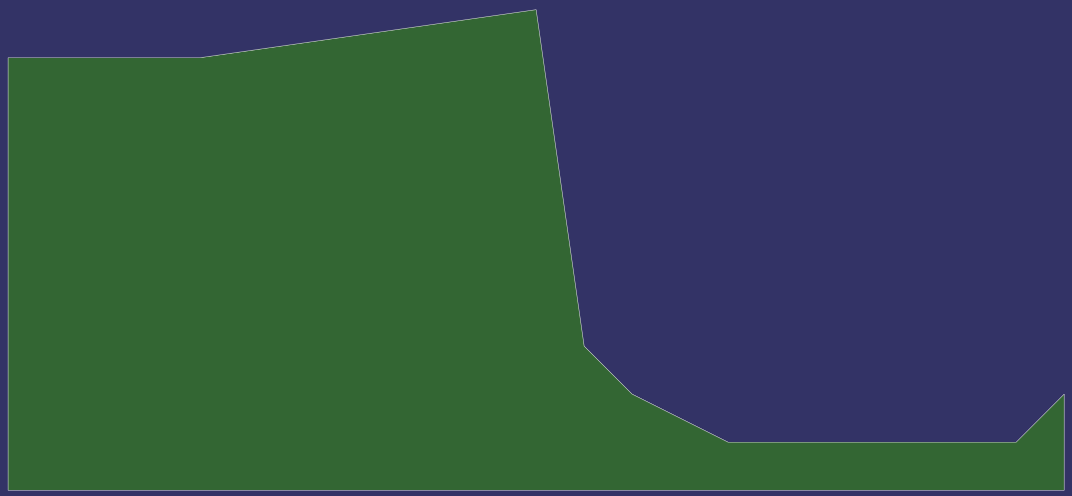
Crater Shader

- Two parts to this shader
  - Height / normal
  - Color
  - Attack each separately, then try to unify



Crater Shader – Height

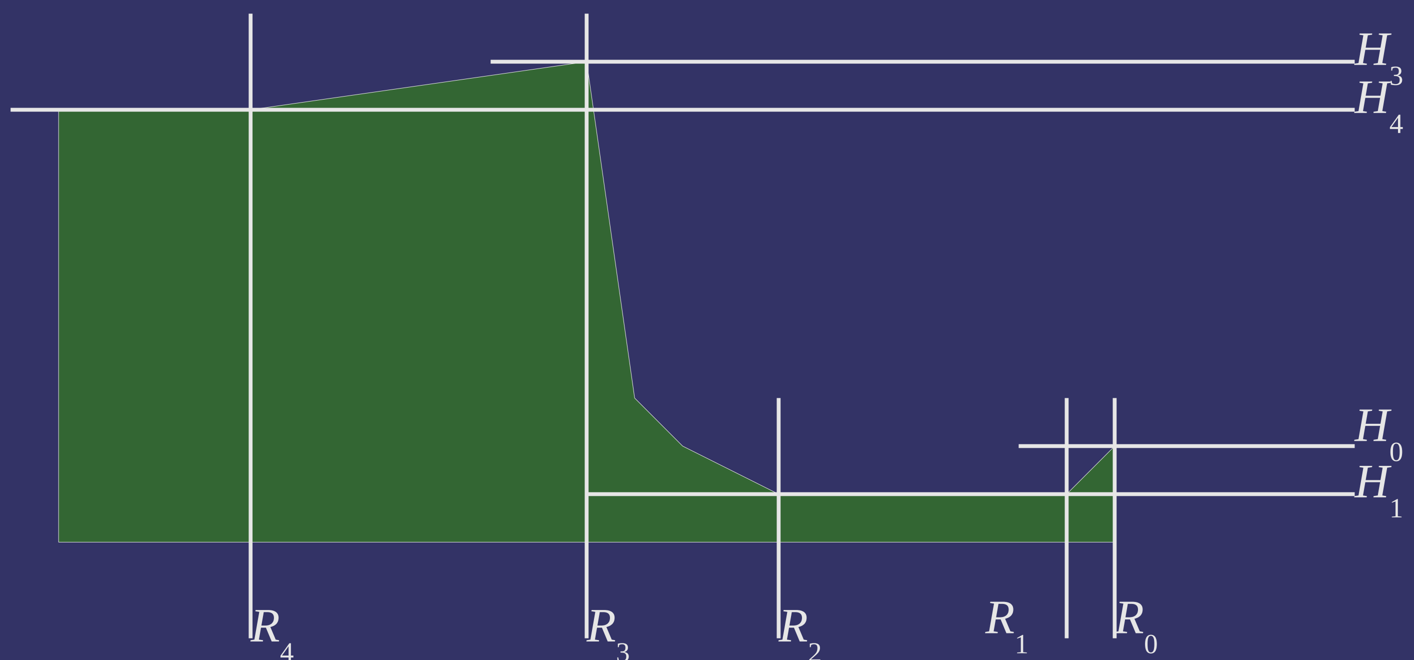
- Craters are generally circular
  - Height varies with distance from center



4-March-2008

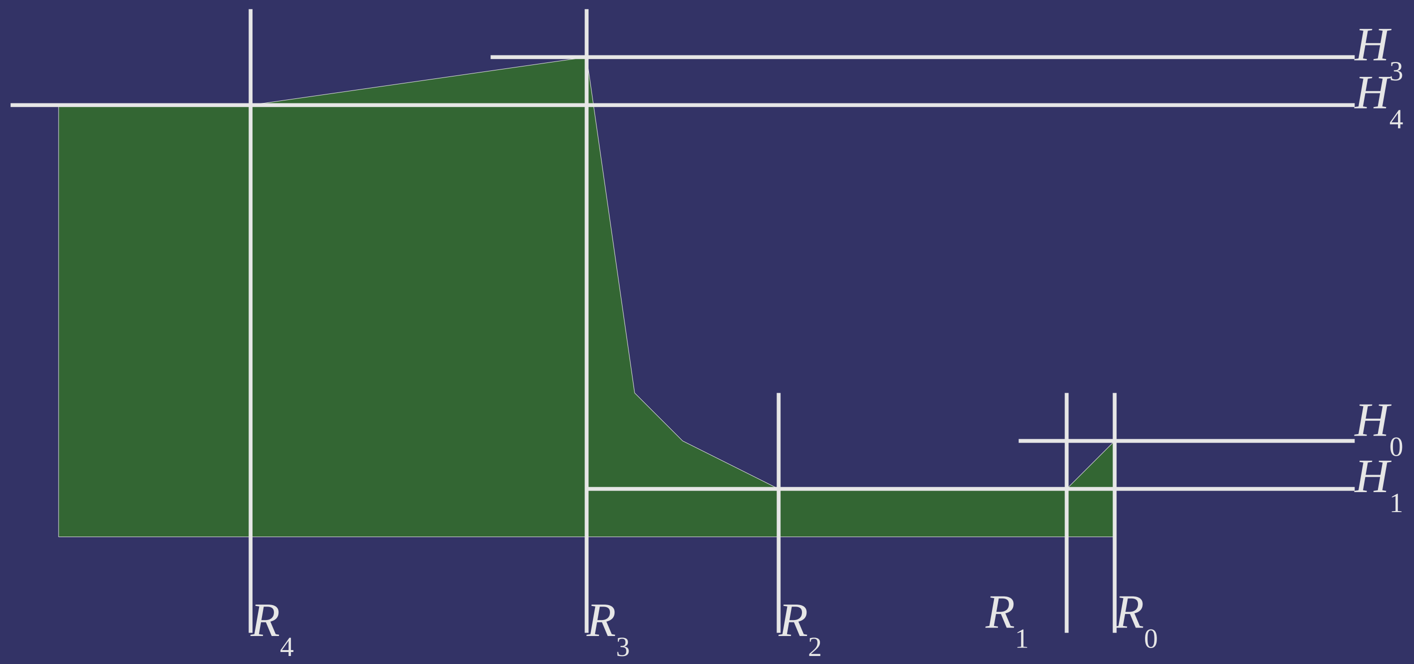
Crater Shader – Height

- Craters are generally circular
  - Height varies with distance from center
  - Associate a height with each distance where there is a change



Crater Shader – Height

- Select an interpolation scheme between each region
  - R_0 to R_1 and R_1 to R_2 could be linear, R_2 to R_3 and R_3 to R_4 could be exponential, etc.



Crater Shader – Height

- In shader:
  - Determine fragment distance from center

```
r = length(position - center);
```

Crater Shader – Height

- In shader:
  - Determine fragment distance from center

```
r = length(position - center);
```

  - Determine which region contains the fragment

```
if (r < crater_parameters[1].x) {
    ...
} else if (r < crater_parameters[2].x) {
    ...
} else ...
```

Crater Shader – Height

- In shader:
  - Determine fragment distance from center

```
r = length(position - center);
```

  - Determine which region contains the fragment

```
if (r < crater_parameters[1].x) {
    ...
} else if (r < crater_parameters[2].x) {
    ...
} else ...
```

  - Determine fragment location in region

```
t = (r - crater_parameters[n].x)
 / (crater_parameters[n+1].x - crater_parameters[n].x);
```

Crater Shader – Height

- In shader:
  - Determine fragment distance from center

```
r = length(position - center);
```

  - Determine which region contains the fragment

```
if (r < crater_parameters[1].x) {
    ...
} else if (r < crater_parameters[2].x) {
    ...
} else ...
```

  - Determine fragment location in region

```
t = (r - crater_parameters[n].x)
 / (crater_parameters[n+1].x - crater_parameters[n].x);
```

  - Perform interpolation

```
h = mix(crater_parameters[n+1].y,
   crater_parameters[n].y, t);
```

Crater Shader – Height

- In shader:
  - Determine fragment distance from center

    ```
    r = length(position - center);
    ```

  - Determine which region contains the fragment

    ```
    if (r < crater_parameters[1].x) {
        ...
    } else if (r < crater_parameters[2].x) {
        ...
    } else ...
    ```

  - Determine fragment location in region

    ```
    t = (r - crater_parameters[n].x)
     / (crater_parameters[n+1].x - crater_parameters[n].x);
    ```

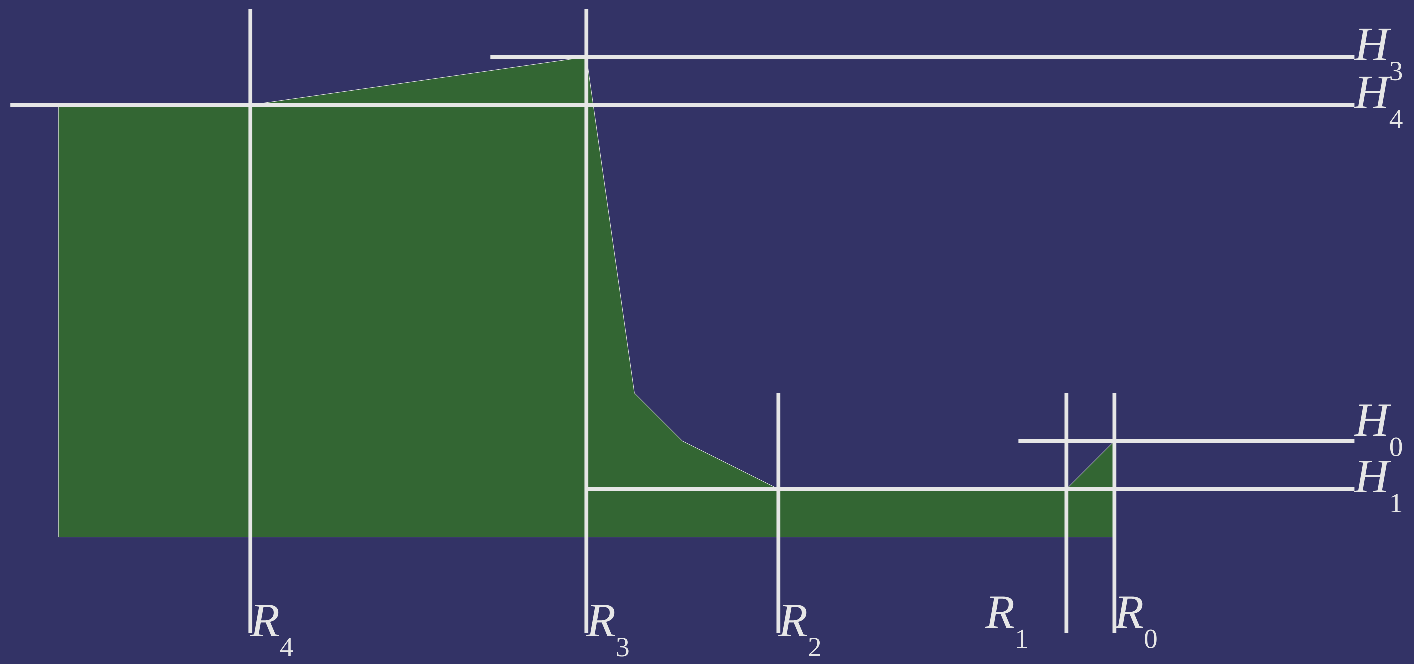
  - Perform interpolation

    ```
    h = mix(crater_parameters[n+1].y,
       crater_parameters[n].y, t);
    ```

  - Write calculated height

Crater Shader – Color

- Color works in a similar manner
  - Use one color inside the crater with alpha set to 1.0
  - Use another color outside the crater
    - Set alpha to 1.0 in “spokes” from crater
    - Falloff to alpha = 0.0 off spokes



Crater Shader – Color

- Selecting crater interior color is trivial
  - If r is less than R_3, use interior color

Crater Shader – Color

- Selecting crater interior color is trivial
  - If r is less than R_3, use interior color
- Selecting spoke color is more complex

Crater Shader – Color

- Selecting crater interior color is trivial
  - If r is less than R_3, use interior color
- Selecting spoke color is more complex
  - Need to know distance from center *and* angle (i.e., polar coordinates)

Crater Shader – Color

- Selecting crater interior color is trivial
  - If r is less than R_3, use interior color
- Selecting spoke color is more complex
  - Need to know distance from center *and* angle (i.e., polar coordinates)
  - Place spokes separated by fixed angles
    - Spokes are determined by a cosine wave in polar coordinates
    - $r_{spoke} = \cos(\alpha \times frequency)$

Crater Shader – Color

- Selecting crater interior color is trivial
  - If r is less than R_3, use interior color
- Selecting spoke color is more complex
  - Need to know distance from center *and* angle (i.e., polar coordinates)
  - Place spokes separated by fixed angles
    - Spokes are determined by a cosine wave in polar coordinates
    - $r_{spoke} = \cos(\alpha \times frequency)$
  - Select random length and thickness for each spoke
    - Noise to the rescue
    - Thickness is determined by raising $(r_{spoke} \times amplitude)$ to a power

References

Ebert, David, et. al., *Texturing and Modeling: A Procedural Approach*, second edition, Morgan-Kaufmann, 1998. pp. 315 – 318.

- This section provided the inspiration for the crater shader.

Next week...

- Depth of field post-process effects
- Discuss final
- Discuss final project

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.