CSE 167

Discussion 10 by Russell Xie
12/05/2018

Agenda

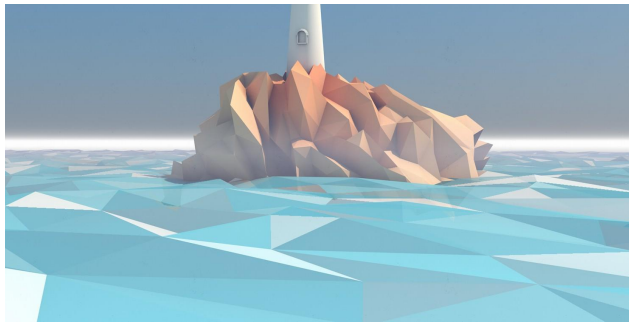
- Final Project Review
- Water Effect with Reflection and Refraction
- Procedurally Generated X
  - Terrain
  - Animated Clouds
  - Plants
- Advanced Collision Detection
- Extra Credit Features
- Presentation and Demo
- Q&A

Final Project Recap

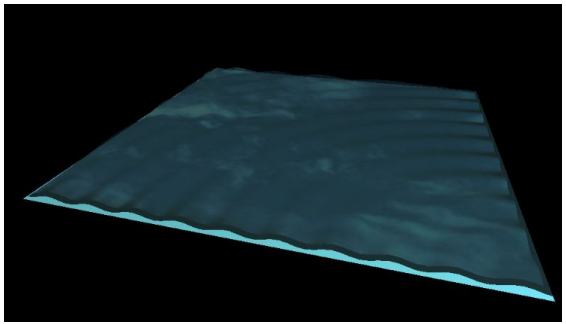
- Link to the Project Page: http://ivl.calit2.net/wiki/index.php/Project5F18
- Project Due Date: Dec 13th, Thursday at 3:00pm
  - Late submission are NOT permitted.
  - Submit to TritonEd and upload the Youtube Video
  - Next Blog due date: Tuesday on Finals week (Dec 11th at night)
- Worth 20% of your grade!
- N medium/hard features required for a team of N people
- Extra Credit features can double count for technical scores!
  - Team of N people must implement N features for full points
- “We will evaluate based on technical and **creative** merits.”

Water Effect

- Double count for “Bezier Patches” and “Water Reflection & Refraction”?
  - Depends on your implementation
  - Demonstrate your Bezier Patches implementation (2 points)
  - Demonstrate your reflection & refraction feature (2 points)



Water Surface without Bezier
(but I love low-poly style!)

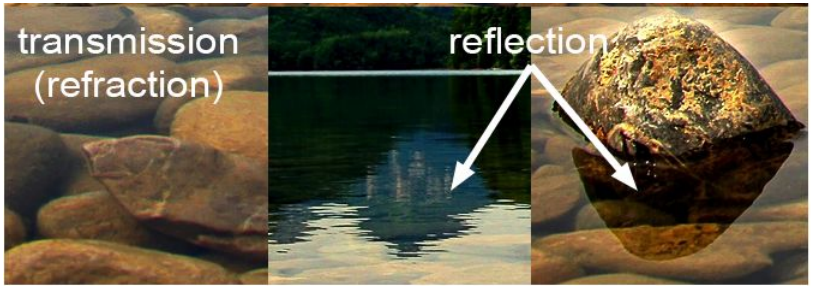


Water Surface without reflection

Water Effect with Reflection and Refraction - FrameBuffer

- FrameBuffer Object is your friend
- Concept of FrameBuffer - Chalkboard Time
- Create FBO for Reflection and Refraction
  - glBindFramebuffer(GL_FRAMEBUFFER, FBO_Reflection);
  - Calc Reflection cam_pos and lookat
  - Render the Scene (without water)
  - Do the same with FBO_Refraction
  - glBindFramebuffer(GL_FRAMEBUFFER, 0);
  - Render Main scene (water with texture)

glFramebufferTexture2D - add texture image to FBO



Water Effect with Reflection and Refraction - Something else to consider

- Generate a simple water surface ?
  - Calc 2D coordinates (x,z), b/c y=0
- Use Depth Buffer?
  - If you need depth effects
  - `vec4 depth1 = texture( tDepth, vUv );`
- glEnable(GL_CLIP_DISTANCE0);
  - Only render geometry above/below the water for reflection/refraction
- Implement the Fresnel Effect:
  - Change how reflective the water is depending on the angle you look at it.

```
float calculateFresnel(){
    vec3 viewVector = normalize(pass_toCameraVector);
    vec3 normal = normalize(pass_normal);
    float refractiveFactor = dot(viewVector, normal);
    refractiveFactor = pow(refractiveFactor, fresnelReflective);
    return clamp(refractiveFactor, 0.0, 1.0);
}
```

```
vec3 finalColour = mix(reflectColour, refractColour, calculateFresnel());
```

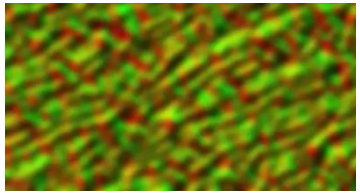
Water Effect with Reflection and Refraction - Something else to consider

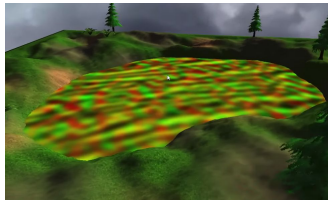
- How do I get the normal of the water surface?
  - One way is to retrieve two neighbor vertices and cross-product
- Add Diffuse and Specular effects for realistic lighting
  - Yes, like in hw2
  -

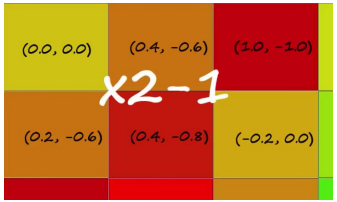
```
color = mix(reflectColor, refractColor, calcFresnel());
color = vec4(vec3(color) * diffuse + specular, 1.0f);
```

- Make the water dynamic/rippling
  - You need a noise function or DuDv map, and adjust vertices in the shader

```
vec2 distortion1 = texture(dudvMap, vec2(textureCoords.x, textureCoords.y)).rg * 2.0 - 1.0;
```







Water Effect with Reflection and Refraction

- Reference:
  - OpenGL Water Tutorial by ThinMatrix: https://www.youtube.com/watch?v=HusvGeEDU_U&list=PLRIWtICgwaX23jiqVByUs0bqhnalNTNZh
  - Reflection and Refraction Shading from Scratchapixel:
  - https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Procedurally Generated X

- The General Idea:
- Construct 3D models or textures algorithmically
  - vs. data-driven modeling
- Usually defined by a small set of data, or rules, that describes the properties of the model
- Often include randomness to add variety
  - E.g.: one algorithm to generate different type of trees using randomness

Procedurally Generated Terrain

- Landscapes are constructed as height fields
- Store height value at each point
- Midpoint Displacement Algorithm and Diamond Square Algorithm

```
Start with single horizontal line segment.
Repeat for sufficiently large number of times
{
  Repeat over each line segment in scene
  {
    Find midpoint of line segment.
    Displace midpoint in Y by random amount.
    Reduce range for random numbers.
  }
}
```

The diamond step: For each square in the array, set the midpoint of that square to be the average of the four corner points plus a random value.

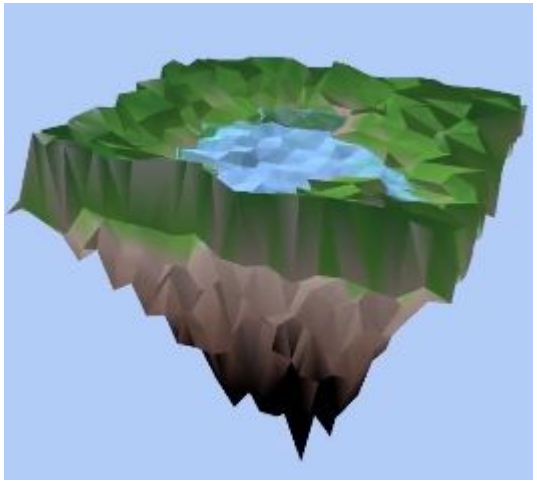
The square step: For each diamond in the array, set the midpoint of that diamond to be the average of the four corner points plus a random value.

At each iteration, the magnitude of the random value should be reduced.

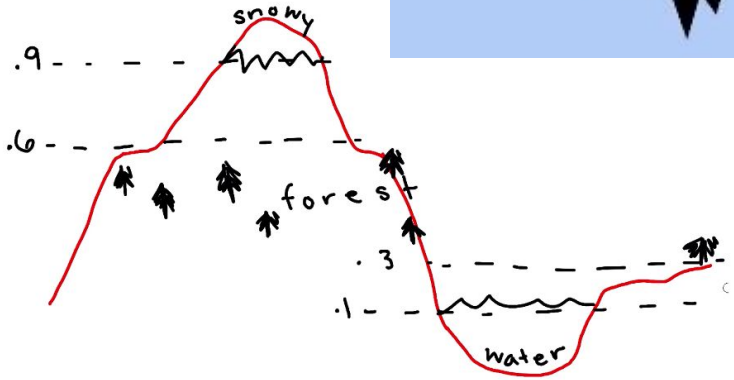
Procedurally Generated Terrain
- Other methods

- Use Perlin Noise
  - To generate a height maps for the terrain
  - perlin_noise(xn,yn) -> deterministic height number
  - Sample Implementation in C++
- Use a combination of wave functions
  - ```
    float generateOffset(float x, float z){
        float radiansX = (x / waveLength + waveTime) * 2.0 * PI;
        float radiansZ = (z / waveLength + waveTime) * 2.0 * PI;
        return waveAmplitude * 0.5 * (sin(radiansZ) + cos(radiansX));
    }
    ```
    ```

Procedurally Generated Terrain - Something else to consider



- Generate Volumetric Terrain
  - Need two different rules
  - Generate the upside and the downside of the terrain
  - Try different sets of configurations to make it look good
- Texture the terrain
  - Define different textures/colors
  - Based on the elevation
  - .0~.1 to be water
  - .3~.4 to be green forests
  - 0.9~1 to be snow
  - etc.



Procedurally Generated Clouds

- To have animated (or perhaps volumetric) clouds, arbitrary shape
- Can be simple but can also be highly complex to achieve realistic effect
- Some reference:
  - A. Webanck et al. / Procedural Cloudscapes
  - Schpok et al / A Real-Time Cloud Modeling, Rendering, and Animation System
  - Petr Man: Generating and Real-Time Rendering of Clouds
- Can achieve similar effects using particle system

Procedural Plants with L-System

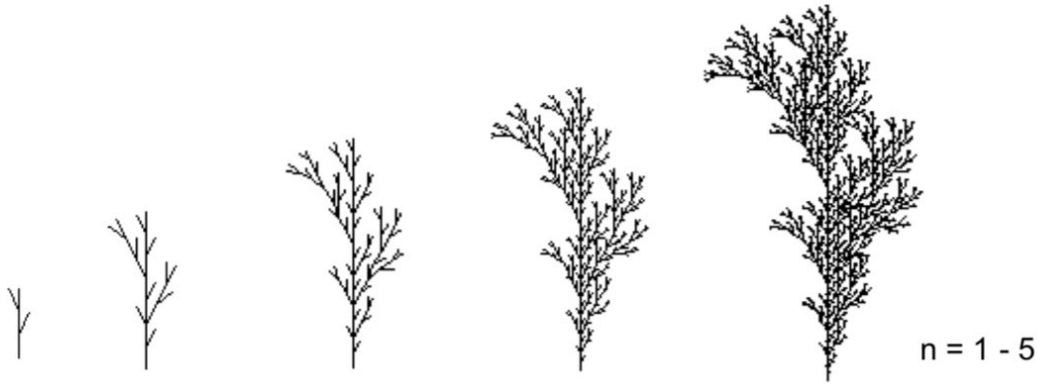
- Grammar = {Variables, constants, initial State, production rules}
- Reference:
- Chalk-board time

root: B

p: B → F

B → F[-B]F[+B][B]

F: move forward
+: turn left
-: turn right
[: Store the current position
]: restore the previous position



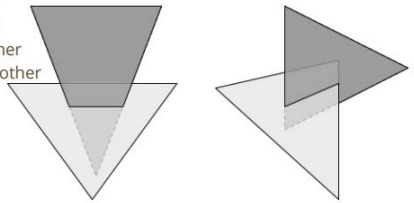
Procedural Plants with L-System

- Requirements: at least 4 variables with parameters
- How to add parameters?
  - Treat each variable as a function
  - Now F[-B]F[+B][B] becomes e.g. F(0.5)[-(60)B]F(1.0)[+(90)B][B]
  - What does this mean?
  - You should use some kind of regulated randomization for parameters
- How to extend to 3D?
  - Add rotational variables: how much to rotate in xz-plane and how much to rotate w.r.t y-axis
  - E.g: F(0.5)[&(90)^(30)B]

Collision Detection with Arbitrary Geometry

- More have been covered in previous discussion slides
- Basic Idea: Recursive Intersection Tests
  - Break object into multiple sections
  - Use small bounding spheres/boxes inside the object
  - Continue to test all triangles inside the bounding sphere/boxes if intersection detected
- Make use of the scene graph engine
- Can be computationally expensive when geometry is very complex
  - Too many triangles to test
  - Not very practical in game engines

- Once you have determined there is an intersection in the small bounding boxes/spheres, perform intersection tests with the triangles inside them
  - This requires you to iterate through the triangles in both objects and testing for intersection on each pair of triangles
- Only 2 possible cases can happen
  - 2 edges of a triangle intersects the other
  - 1 edge of each triangle intersects the other



Extra Credit Features

- Water Effect: Covered
- SSAO: https://learnopengl.com/Advanced-Lighting/SSAO
- Motion Blur: https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch27.html
- DOF Effect: Often use Bokeh Blur; Be able to change aperture/focus
  - Example that I found useful: http://artmartinsh.blogspot.com/2010/02/glsl-lens-blur-filter-with-bokeh.html
- Make use of FBO's depth Buffer:

Presentation and Demo - Logistics

- Agenda for Thursday on Finals Week (Dec 13th)
  - 3pm - 4pm: Screening of videos made by each team (CENTER 113)
  - 4pm - 5pm: Group A science-fair style demos in the CSE Lab (B260/B270 or B210/B220)
  - 5pm - 6pm: Group B science-fair style demos in the CSE Lab (B260/B270 or B210/B220)
- Everyone needs to show up at 3pm for the video screening
- Graders will all be trying out your application during demos.
  - Be sure to practice demoing
  - Grades will not be decided on the spot
- Try out projects from other groups as well!

Presentation and Demo - How to give a good demo

- A good/bad demo experience can sometime affect user's impression when trying out your application!
- Note that the "subjective" part your grade may depend on your demo

- Tells us clearly what technical features you implemented. Is it complete?
- Do you have other efforts you want to share about your project?
- Do your features have a toggle switch?
- Do your EC features have visual debugging aids?
- Can a grader/student try out your demo?

Questions?

Thanks for choosing CSE 167! 🎉
Good Luck on the Final Project!