CIS565 Final Project: Procedural Terrain Generation with Vulkan

Mauricio Mutai and Rudraksha Shah

Technical Overview

- The primary goal of the project was to implement the following pipelines and measure their performance impact on a tessellation-heavy procedurally generated terrain:
  - **Forward pipeline.**
    - Classic forward pipeline with separate shaders for geometry and skybox
  - **Deferred pipeline.**
    - Classic deferred pipeline where first pass generates G-buffer data, which is shaded by second pass
  - **Visibility pipeline**.
    - A new take on the deferred pipeline as proposed in this JCGT paper by Burns and Hunt. It has a Visibility buffer that is similar to the G-buffer but instead of the traditional G-buffers for normals, position and albedo the Visibility buffer only stores the triangle Index and Instance ID.

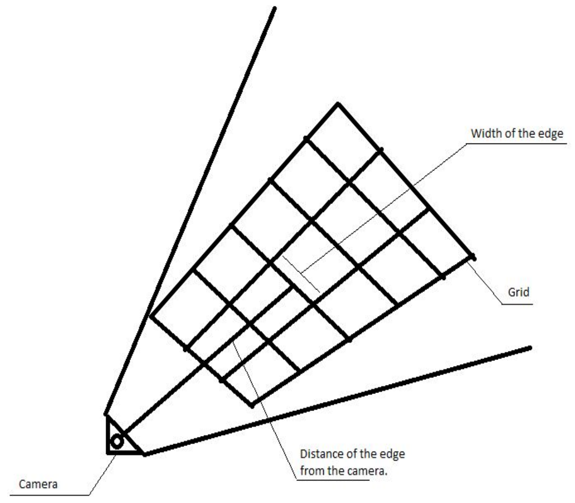
Technical Overview (continued)

- The paper suggests a way to implement the visibility pipeline for tessellated geometry but does not follow through with any implementation.
- For our project we have implemented the visibility pipeline for tessellated geometry. And in the V-buffer instead of storing triangle index and index ID we store the terrains XZ position and the UV values.
- These values are used in the second rendering stage to re-produce normals, generate shadows and shading + texturing.

Features

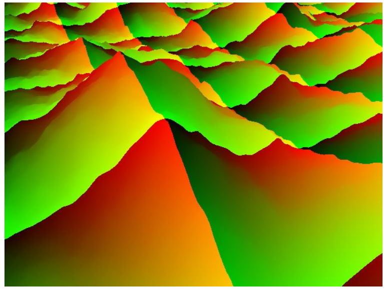
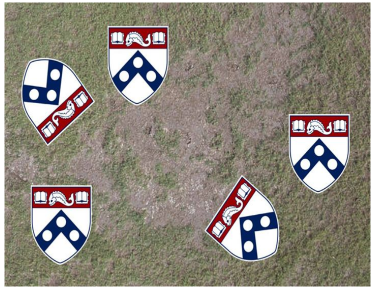
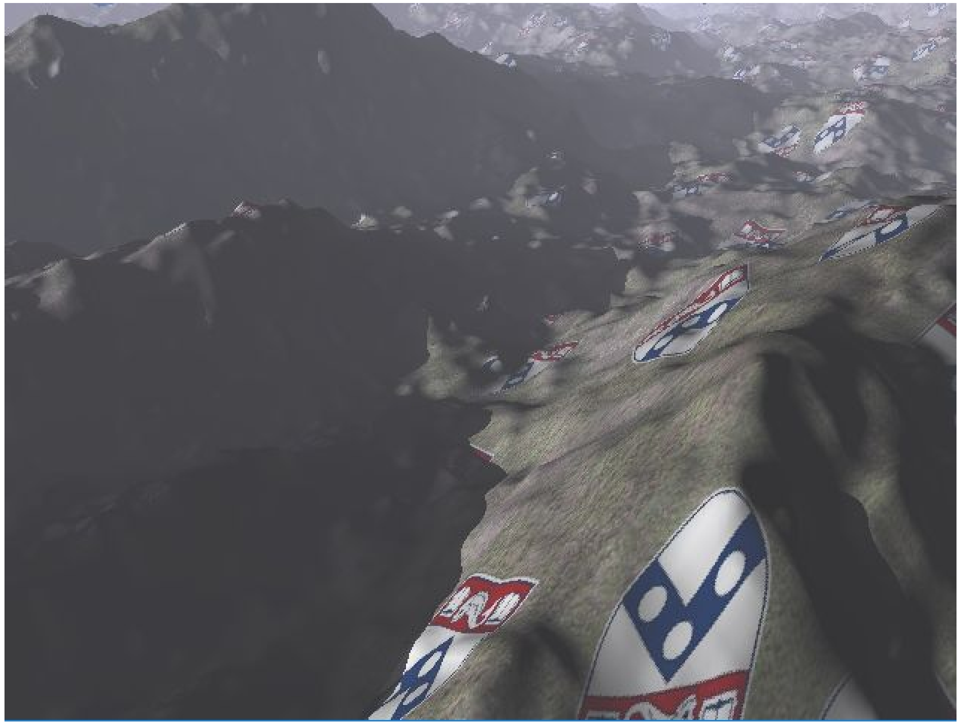
Dynamic Level of Detail

- The terrain is composed of cells, which are quads on the XZ plane.
- The offsets for each cell in the grid is sent to the compute shader which calculates the distance of each edge from the camera and uses it to determine each edge's LOD (tessellation level).
- Provides a significant performance boost to each pipeline.



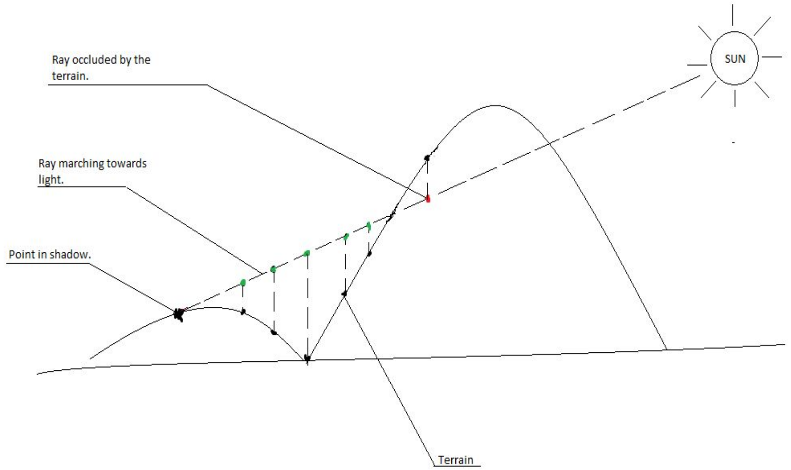
Texture mapping

- Each point on the terrain is represented by a UV value between [0,1] which is used to sample the texture for texture mapping.

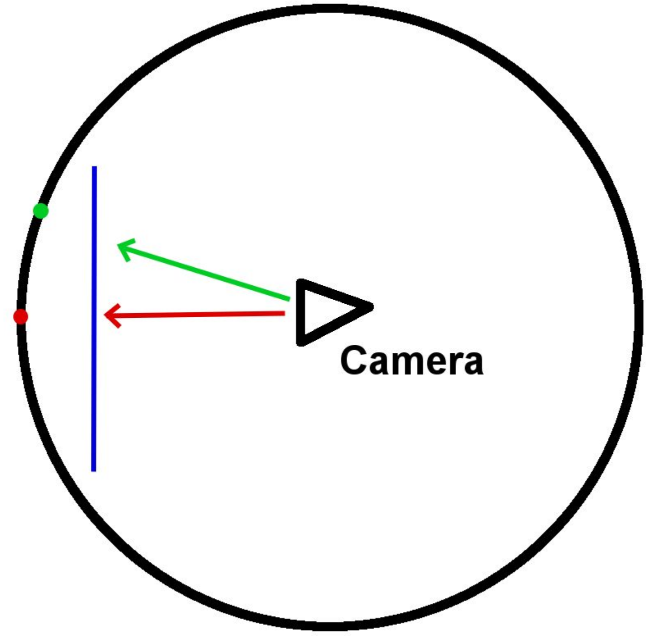
Shadows

- For determining whether a point on the terrain is in shadow or not we ray-march from the point towards the light.
- At every point we sample the noise to determine the height of the terrain and if the height of the point along the ray is below the terrain, the point is in shadow.
- This is the primary bottleneck for each pipeline.



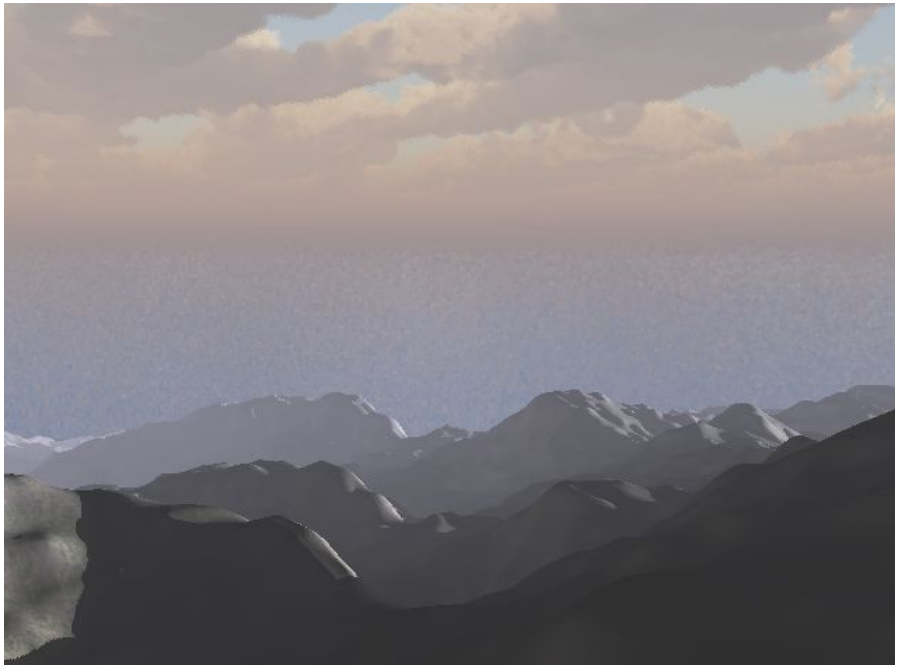
Skybox & Sun

- For each screen-space fragment, we can obtain a world-space direction from the camera to the fragment.
- Based on this direction, we can sample a skybox texture.
- By defining a "sun direction", we can draw a sun in the fragments where the angle between the sun direction and the fragment's direction is below a certain threshold.



Height Based Fog

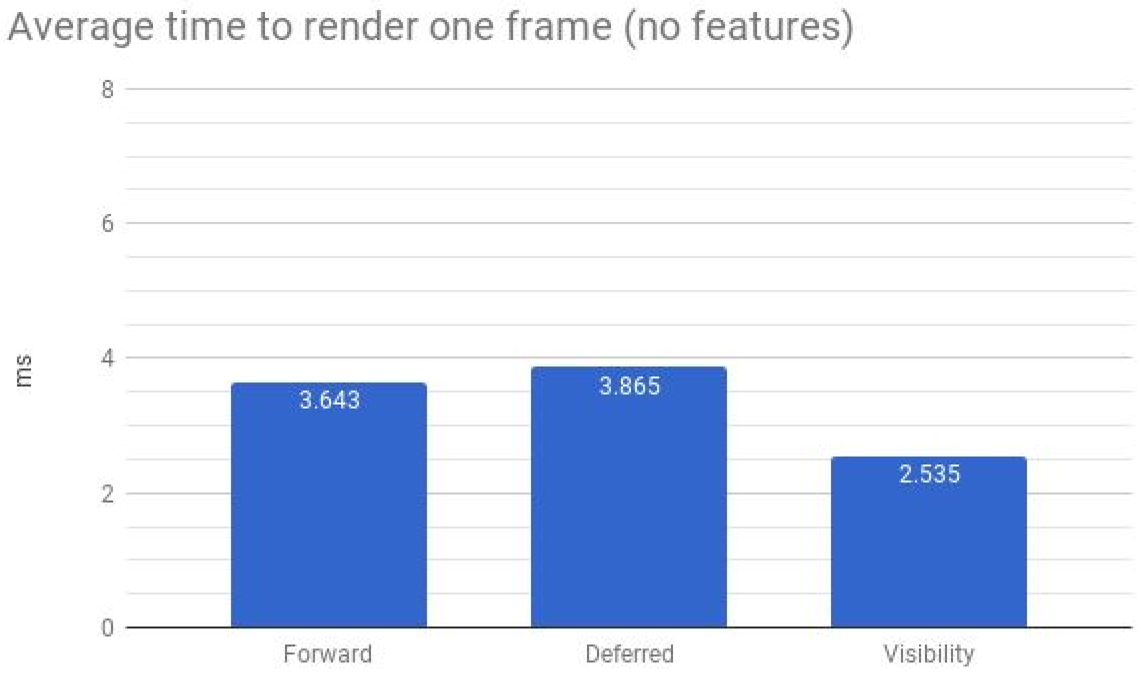
- The fog is generated by using a parametric equation.
- The distance of the surface point from the camera is fed to this equation that generates a value between [0, 1] which is used to blend the final color with the fog color.



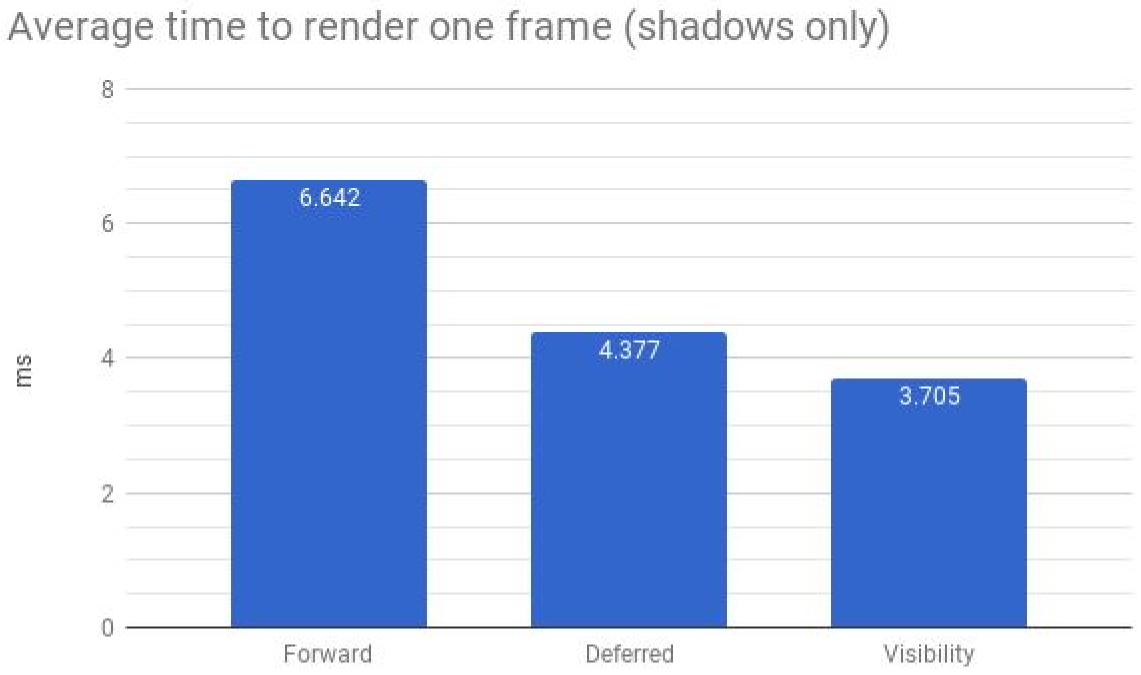
```
fogFactor = exp(-rayOrigin.y) * (1.0 - exp(-distance * rayDirection.y)) / rayDir.y
```

Video

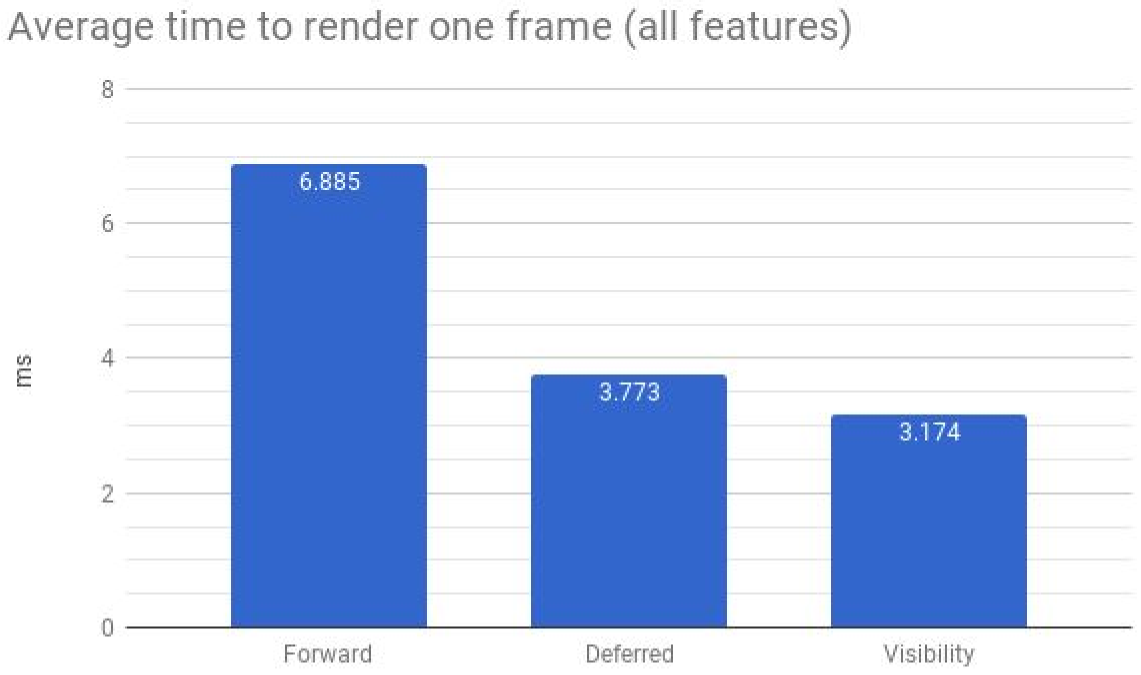
Performance Analysis



Performance Analysis (continued)



Performance Analysis (continued)



Shortcomings

- Transparency: would require significant modifications to visibility pipeline.
- Ray-marched shadows: per-fragment + computationally heavy => slow.
- Noise sampling: Sampling the noise at every step during ray-marching could be a possible point to investigate/optimize. By using more efficient texture to store the height values for the terrain (height maps) we can reduce the cost of ray-marching shadows.

Thank you!