SWOOP VR

A WEB-BASED MESH VIEWER

Created by Chris Foster / @chrisfosterelli

DEVELOPMENT GOALS

Both a research and development project for:

A functional server-side application that can handle the upload of 3D object files and represent these to a client-side application via an API that exposes smaller, more efficient chunks.

A functional client-side browser application that improves on the previous iteration in key ways and utilizes the new server-side API.

PROBLEM BACKGROUND

- 3D scene data from drone scans can be hundreds of MB
- It isn't efficient to send this to the browser
- Most current sites simply "downgrade" the quality

PROPOSED SOLUTION

- Split the 3D scene into separate "chunks"
- Generate different Level of Detail (LOD) for each chunk
- Client would intelligently request quality and chunk

CLIENT OVERVIEW

- Written in Javascript
- Utilizes three.js WebGL library

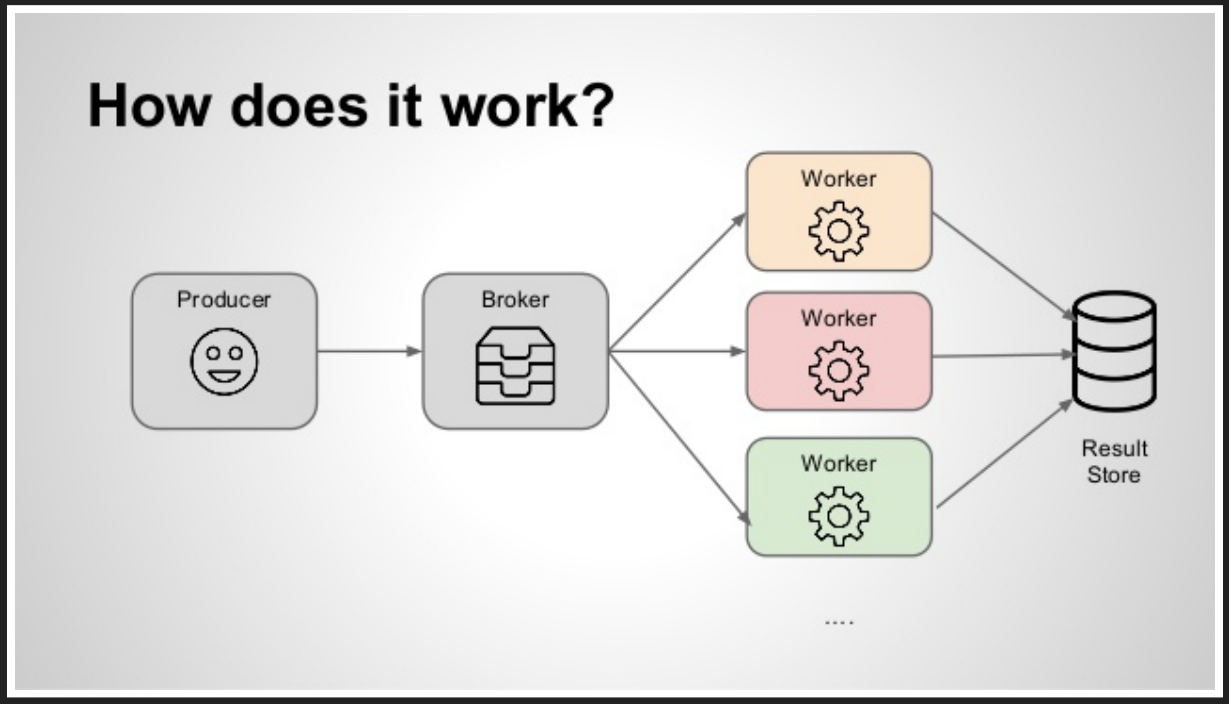
SERVER OVERVIEW

- Written in Python
- Utilizes Flask web framework

DATABASE OVERVIEW

- Uses RethinkDB for metadata storage
- Uses Redis for batch job queueing

BATCH PROCESSING



- Server sends items to a Celery worker
- Worker queue is managed with Redis
- Workers return results back to Flask

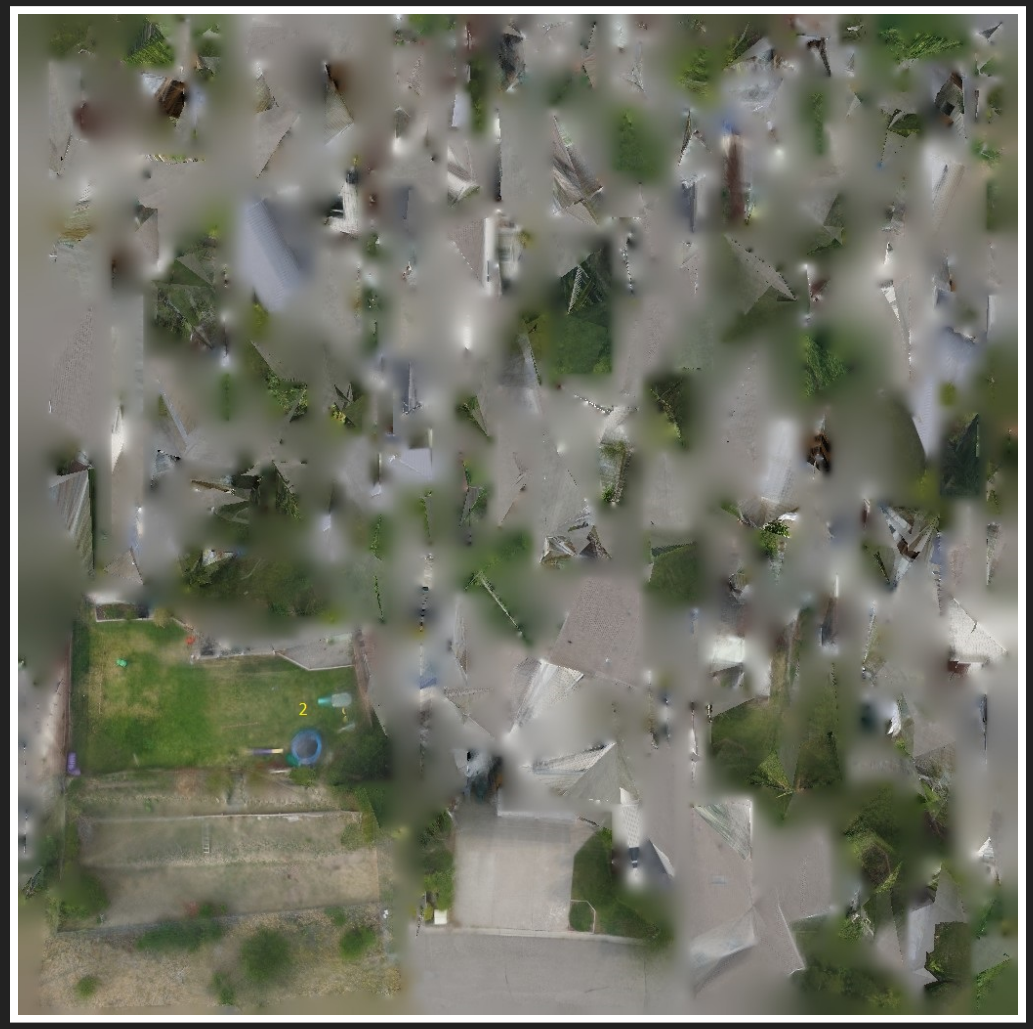
WORKER OVERVIEW

- Our worker manipulates and deals with 3D mesh files
- We tried to find an existing library to do this
- Nothing exists! Our code parses Wavefront directly
- In retrospect, this has caused problems...

WAVEFRONT FORMAT

- Three files: scene.OBJ, scene.MTL, scene.JPG
- These three files coordinate to each other
- Difficult to manage programmatically

TEXTURE FILE



MATERIAL FILE

- Simple data file that provides metadata about the texture
- Our script doesn't need to worry about this a lot

```
newmtl 02
Ka  1.0 1.0 1.0
Kd  1.0 1.0 1.0
Ks  0.0 0.0 0.0
d  1.0
Ns  0.0
illum 0
map_Kd 02.jpg
```

MESH FILE

- Our mesh files contain three types of entries
- `v` defines a 3D vertex
- `vt` defines a texture coordinate in the material
- `f` defines a face using offset coordinates of the mesh file
- This makes editing hard

```
mtllib 02.mtl
usemtl 02
...
v 0.073897 -10.633666 5.916892
v -0.078425 -11.680308 6.108032
v -1.933728 -11.567311 6.207512
...
vt 0.031250 0.526257
vt 0.055772 0.567377
vt 0.076542 0.429486
...
f 95/59 96/60 99/62
f 96/60 102/63 99/62
f 102/54 96/57 109/55
...
```

CHUNKING

Chunking separates a 3D mesh into pieces:

1. Load all data into arrays
2. Take groups of 1000 faces
3. For each face:
    1. Find the matching vertices
    2. Find the matching vertex textures
    3. Write them into new arrays
    4. Rewrite face to use new indices
4. Write output to new chunk file

DECIMATION

Decimation simplifies a 3D mesh:

1. Find candidate vertices based on a condition
2. Delete all the candidate vertices
3. Delete all faces that use the candidate vertices
4. Make new faces using Delaunay Triangulation

WAVEFRONT DRAWBACKS

- Removing a vertex requires recalculating 50% of faces
- Removing a vertex requires recalculating vertex textures
- Removing a vertex texture requires recalculating all faces
- Generating a new face is easy, associating texture is not
- All publically available algorithms ignore texture details
- Wavefront supports **a lot** more than we support
- Wavefront faces are not always ordered, as we assume

Because of this, *our decimation algorithm didn't work well*

ALTERNATIVE SOLUTION

- Most decimation algorithms are hundreds of lines...
- Most 3D tools also have support for working with meshes
- *Rolling our own* isn't a good solution, but no library exists!
- Proposal: headless Blender renderer

DEMO

PROJECT SUMMARY

- Chunk generation for a given scene
- Intelligent worker processing of 3D scenes
- Basic web portal for account/project management
- Client application for rendering 3D scenes with chunks
- Researched and prototyped decimation algorithm for LOD
- Researched and determined best future implementation path

THE END

- Chris Foster
- Thompson Rivers University