HuBMAP (Human BioMolecular Atlas Program)

Jyotiprakash Panda
Pritish Thombare

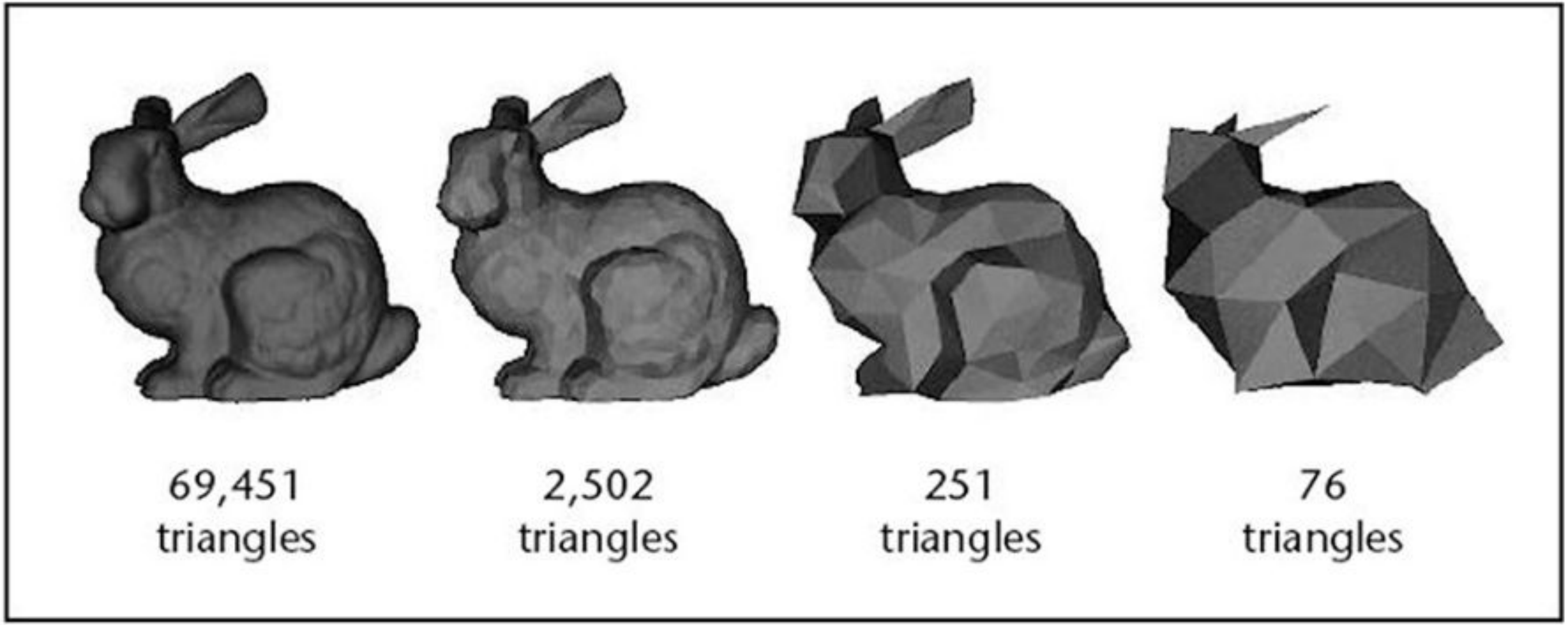
Fall 23

Level of Detail

- An important tool to maintain interactivity
- Focuses on performance tradeoff
- Aims to solve the problem of rendering complex Geometric datasets at interactive rates
- Also called Polygon simplification, geometric simplification, mesh reduction

Applications of Level of Detail (LOD) in 3D Modeling

- Medical Imaging of Anatomical models
- Computer Graphics & Gaming
- Virtual Reality (VR) & Augmented Reality (AR)
- Simulation & Training
- Geographic Information Systems (GIS)
- Navigation & Maps
- Architectural Visualization


69,451
triangles
2,502
triangles
251
triangles
76
triangles

Level of Detail Framework - Discrete LOD

- Description
  - Preprocessed LODs rendered according to the distance of the object.
- Advantages
  - Decouples simplification and rendering.
  - Rendering is much faster.
- Disadvantages
  - Not suited for drastic simplification.
  - Causes visual popping during transitions.

Level of Detail Framework - Continuous LOD

- Description
  - Uses a data structure from the desired level of details can be extracted at runtime.
- Advantages
  - Better granularity and smoother transitions
  - Supports progressive transmission
  - Leads to view dependent LOD

Choosing LODs

- Need to choose correct LODs to maintain constant frame rates
- One solution for this is the Predictive LOD selection
- For each LOD, we should estimate:
    - Cost or the rendering time
        - Number of triangles
        - How large it is on the screen
        - Lighting and texturing etc.
    - Benefit or the importance to the image
        - Distance
        - Size
        - Eccentricity
        - Velocity

Proposed Work

In the proposed project, our objective was to streamline the process of image simplification using the CGAL library.

To achieve this, we initially converted the images from the GLB format to the OFF format. This conversion was essential because the CGAL library requires images in the OFF format to perform the simplification process effectively.

By doing so, we ensured compatibility with CGAL's tools and algorithms, enabling us to efficiently simplify the images as intended.

Notes

- HuBMAP: https://hubmapconsortium.github.io/ccf/pages/ccf-3d-reference-library.html
- https://github.com/hubmapconsortium/ccf-tissue-block-annotation/tree/main
-

Action items

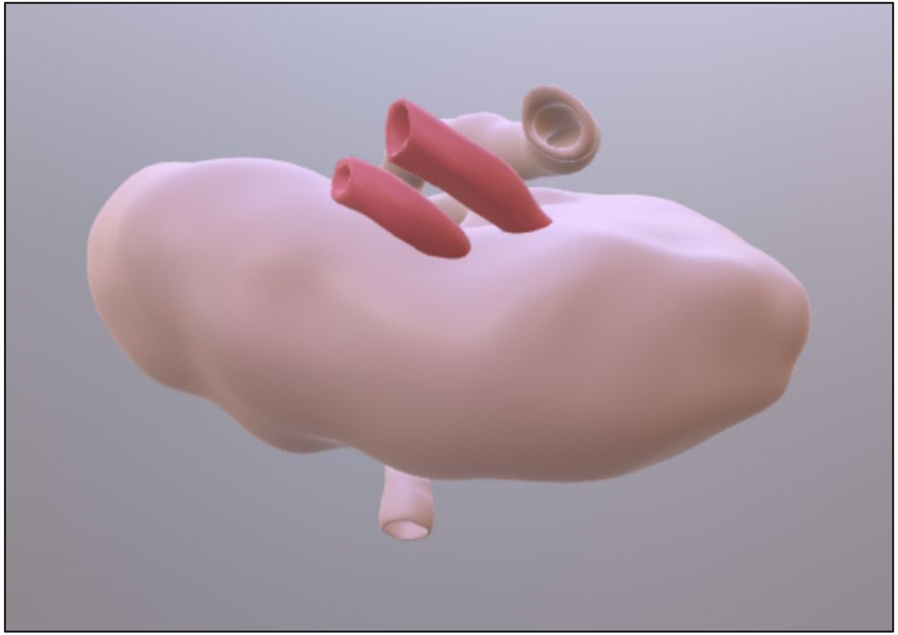
- [ ] Parse GLB files into different meshes (.off) using Python
    - [ ] download the 3D models in GLB format.
      https://github.com/hubmapconsortium/ccf-releases/tree/main/v1.3/models
      In each single GLB file, there are multiple meshes. Each mesh will be saved in a single OFF file. A GLB file -> create a folder with multiple OFF files.
    - [ ] Babylon.js https://sandbox.babylonjs.com/
    - [ ] Parse them using Python package pygltflib. Please read the reference of this package.
        - [ ] 1. https://pypi.org/project/pygltflib/
        - [ ] 2.https://github.com/hubmapconsortium/ccf-tissue-block-annotation/blob/main/glbparser/glb_parser.py
- [ ] CGAL: Triangulated mesh simplification
  https://doc.cgal.org/latest/Surface_mesh_simplification/index.html
- [ ] LOD level of detail
- [ ] wrap the off files to GLB.

CGAL installation install 1 - 6
Scripts
glb_parser: (one) glb to (multiple) off
glb_converter: (one) off to (one) glb

GLB Images

This is an example of GLB image of Lymph node .

Our aim is to convert these images into OFF format on which we can use the CGAL library for mesh simplification.



Installing packages

The following packages were installed for creating the environment:

Installed Virtual box and setup Ubuntu in it.

Installed dependencies like Boost, GMP, MPFR, CGAL

Installed many Python packages like numpy, pygltflib

Boost

```
sudo apt-get update
sudo apt-get install libboost-all-dev
```

GMP

```
sudo apt-get install libgmp-dev
```

MPFR

```
sudo apt-get install libmpfr-dev
```

CGAL

```
sudo apt-get install libcgal-dev
```

Eigen3

```
sudo apt install libeigen3-dev
```

cpprestsdk

```
sudo apt-get install libcpprest-dev
```

Python library [3]:

pygltflib

```
pip install pygltflib
```

GLB Parser

This file converts the GLB files to OFF files on which we will be using CGAL library to do mesh simplification.

```
def glb_plain_parser(input_dir, organ: str, output_dir):

    file = os.path.join(input_dir, organ + '.glb')
    data_type_dict = {5121: 'uint8', 5123: 'uint16', 5125: 'uint32', 5126: 'float32'}
    number_of_components = {'SCALAR': 1, 'VEC2': 2, 'VEC3': 3, 'VEC4': 4, 'MAT2': 4, 'MAT3': 9, 'MAT4': 16}

    glb = GLTF2.load(file)
    binary_blob = glb.binary_blob()
    output_organ_dir = os.path.join(output_dir, organ)

    for mesh in glb.meshes:

        mesh_name = mesh.name

        triangles_accessor = glb.accessors[mesh.primitives[0].indices]
        triangles_buffer_view = glb.bufferViews[triangles_accessor.bufferView]
        dtype = data_type_dict[triangles_accessor.componentType]

        triangles = np.frombuffer(
            binary_blob[triangles_buffer_view.byteOffset + triangles_accessor.byteOffset:
                        triangles_buffer_view.byteOffset + triangles_buffer_view.byteLength],
            dtype=dtype,
            count=triangles_accessor.count,
        ).reshape((-1, 3))

        points_accessor = glb.accessors[mesh.primitives[0].attributes.POSITION]
        points_buffer_view = glb.bufferViews[points_accessor.bufferView]
        dtype = data_type_dict[points_accessor.componentType]

        points = np.frombuffer(
            binary_blob[points_buffer_view.byteOffset + points_accessor.byteOffset:
            points_buffer_view.byteOffset + points_buffer_view.byteLength],
            dtype=dtype,
            count=points_accessor.count * 3,
        ).reshape((-1, 3))
-- INSERT --
```

CMake file

We put the project configurations and build requirements into the cmake file.

```
cmake_minimum_required(VERSION 3.1...3.14)
project(CgalProject)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED True)

find_package(CGAL REQUIRED)

add_executable(CgalProject meshSimplification.cpp)
target_link_libraries(CgalProject CGAL::CGAL)
```

CGAL

Learnt about the CGAL library.

Understood the various techniques CGAL used to work on mesh simplification.

The Edge collapse operation

The Cost strategy it uses to perform the operation i.e. Lindstrom-Turk Strategy

```
#include <CGAL/Simple_cartesian.h>
#include <CGAL/Surface_mesh.h>

#include <CGAL/Surface_mesh_simplification/edge_collapse.h>
#include <CGAL/Surface_mesh_simplification/Policies/Edge_collapse/Edge_count_ratio_stop_predicate.h>

#include <chrono>
#include <fstream>
#include <iostream>

typedef CGAL::Simple_cartesian<double>               Kernel;
typedef Kernel::Point_3                              Point_3;
typedef CGAL::Surface_mesh<Point_3>                  Surface_mesh;

namespace SMS = CGAL::Surface_mesh_simplification;

int main(int argc, char** argv)
{
  Surface_mesh surface_mesh;
  const std::string filename = (argc > 1) ? argv[1] : CGAL::data_file_path("meshes/cube-meshed.off");
  std::ifstream is(filename);
  if(!is || !(is >> surface_mesh))
  {
    std::cerr << "Failed to read input mesh: " << filename << std::endl;
    return EXIT_FAILURE;
  }

  if(!CGAL::is_triangle_mesh(surface_mesh))
  {
    std::cerr << "Input geometry is not triangulated." << std::endl;
    return EXIT_FAILURE;
  }
```

Triangulated Mesh Simplification

This is the main code which performs the mesh simplification on the image files.

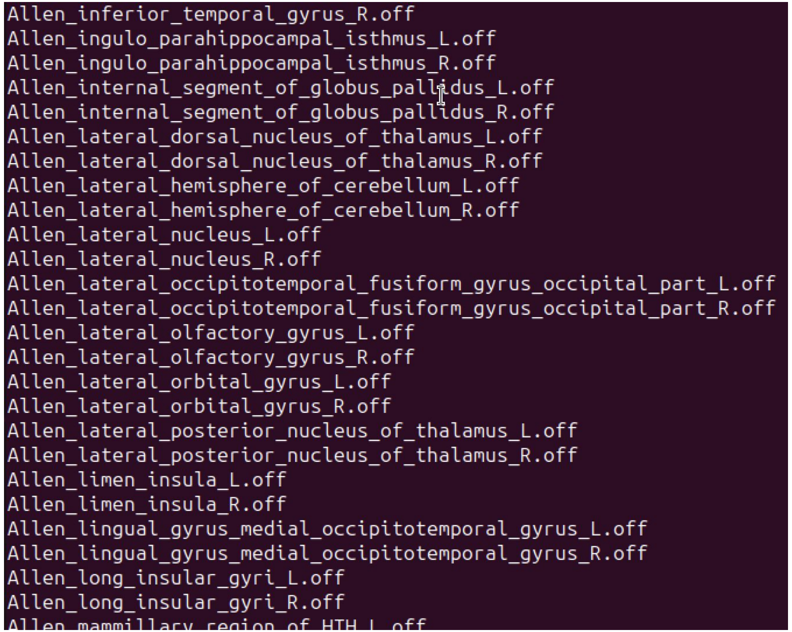
On the command line we can put the source and destination files along with the stop ratio (amount of simplification we want in our image).

```
void triangulatedMeshSimplification(string inputSubDirectory, string inputFilename, string outputDirectory, double stop_){
  Surface_mesh surface_mesh;
  const std::string filename = CGAL::data_file_path(inputFilename);
  std::ifstream is(filename);
  if(!is || !(is >> surface_mesh))
  {
    std::cerr << "Failed to read input mesh: " << filename << std::endl;
  }
  if(!CGAL::is_triangle_mesh(surface_mesh))
  {
    std::cerr << "Input geometry is not triangulated." << std::endl;
  }
  std::chrono::steady_clock::time_point start_time = std::chrono::steady_clock::now();
  // In this example, the simplification stops when the number of undirected edges
  // drops below 10% of the initial count
  double stop_ratio = stop_;
  SMS::Count_ratio_stop_predicate<Surface_mesh> stop(stop_ratio);
  int r = SMS::edge_collapse(surface_mesh, stop);
  std::chrono::steady_clock::time_point end_time = std::chrono::steady_clock::now();
  std::cout << "\nFinished!\n" << r << " edges removed.\n" << surface_mesh.number_of_edges() << " final edges.\n";
  std::cout << "Time elapsed: " << std::chrono::duration_cast<std::chrono::milliseconds>(end_time - start_time).count() << "ms" << std::endl;
  saveSimplifiedMesh(inputSubDirectory, inputFilename, outputDirectory, surface_mesh);
  //CGAL::IO::write_polygon_mesh(outputFilename, surface_mesh, CGAL::parameters::stream_precision(17));

}
```

Simplified Output Results

After simplification using the CGAL Library these results were rendered.



Future Work

Adopting Python for Mesh Simplification
Transition: Shift from C++ CGAL to Python libraries for image processing and mesh simplification, aiming for increased flexibility and ease of use.

Integrating CGAL and Blender:
Blender's Role: Implement Blender for direct GLB processing and creating various LODs, enhancing the mesh simplification workflow.

Workflow Setup:
Initial Action: Install and configure Blender to integrate with our existing pipeline, leveraging its advanced capabilities for improved processing efficiency.