01.110: Computational Fabrication Summer 2017

Programming Assignment 1: Voxelization

Due May 30th at 11:55pm.

In this assignment you will implement a voxelizer. Your program will take, as input, a triangulated, manifold surface mesh and the dimensions of the requested voxelization. As output your program will generate a voxelization, saved to an .obj file for display. Provided with this assignment is a code stub which can be compiled on Mac OSX, Linux or Microsoft Windows. These instructions assume that you are running the code under OSX or Linux.

This voxelizer will be based on ray casting. We will partition the space containing an input surface into a regular grid of specified dimension. Each cubic cell in this grid is a potential voxel in the voxelization. Your code will determine whether a voxel lies inside or outside the surface mesh. For each voxel in the grid you will cast a ray. As discussed in class, counting the number of times this ray intersects the object surface will suffice as an inside/outside test.

In this assignment you will be responsible for implementing the following components of the voxelizer:

1. A loop over all voxels in the regular grid
2. A ray-triangle intersection test
3. An inside-outside test for all voxels that uses a ray cast in a single direction

The remainder of this document is organized as follows:

1. Getting Started
2. Starter Code and Implementation Notes
3. Extra Credit
4. Submission Instructions

1 Getting Started

1.1 Building Project Files

We use CMake to ensure that assignments can be built on multiple platforms. Here we outline how to use CMake to generate platform specific build files. We use `<WORKINGDIR>` to refer to the root directory of the assignment source code tree (i.e the directory that contains this PDF).

1. Download CMake from http://www.cmake.org
2. Open a terminal window and change directory to `<WORKINGDIR>`
3. Create the directory `<WORKINGDIR>/build`, change to the build directory
4. In `<WORKINGDIR>/build/`, run: `cmake .. -G'Unix Makefiles'`
5. In `<WORKINGDIR>/build/`, run: `ccmake ..`
6. Under CMAKE_BUILD_TYPE enter `Debug`
7. Press: 'c', then 'g'
8. You should now have a generated makefile in `<WORKINGDIR>/build`, type `make` to build the voxelizer

You can build other project types with CMake by changing the `-G` option in step 4. **To generate Xcode projects** use `cmake .. -G'Xcode'`. **On windows** use the CMake GUI to generate build projects.

1.2 Running the Sample Solution

We provide sample solutions for a variety of platforms in the `<WORKINGDIR>/samples`. We provide sample triangle meshes in the `<WORKINGDIR>/data` directory. **Run the code using:**

`voxelizer pathToInput.obj pathtoOutput.obj`.

e.g. From `<WORKINGDIR>/samples/OSX`:

```
voxelizer ../../data/sphere/Sphere.obj ../../data/sphere/SphereVoxels.obj
```

To see the results open the output file in Meshlab (http://meshlab.sourceforge.net).

2 Starter Code and Implementation Notes

We provide starter code for the assignment. Critical sections of the code have been removed, these are clearly labelled with /********* ASSIGNMENT *********/, followed by appropriate comments. You will be responsible for implementing (at minimum) the components specified previously. The starter code provides the following features

1. A Vec3 class with associated dot product, cross product, addition and subtraction operations (CompFab.h)
2. A Ray class (CompFab.h)
3. A Triangle class (CompFab.h)
4. A VoxelGrid class which contains an *isInside* method for labeling voxels (CompFab.h)
5. An input method for reading surface meshes stored as .obj files and an output method for saving your voxelizations (main.cpp)

The starter code can load an object and will automatically initialize the VoxelGrid using the dimensions you specify. Your sole task is to label voxels as inside our outside the mesh.

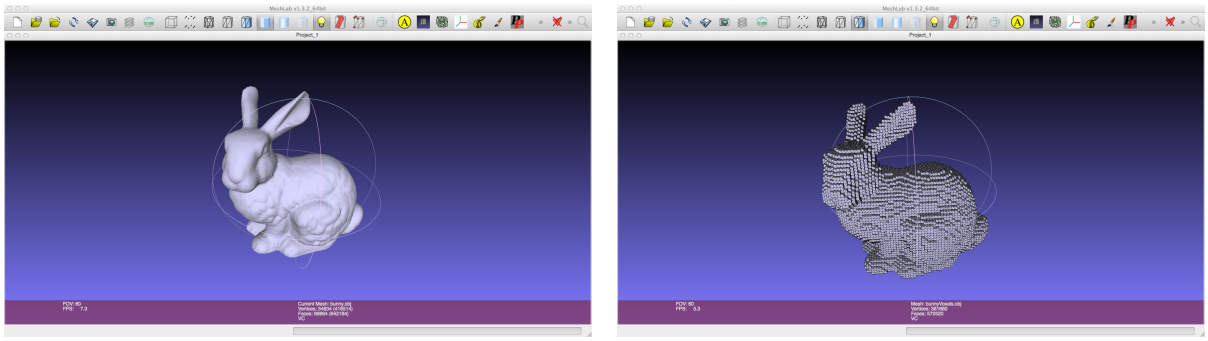

Figure 1: Voxelizing a bunny mesh using the sample solution.

3 Extra Credit

There are two potential features to implement for extra credit:

1. Casting a ray in a single direction makes your voxelizer less robust to errors in meshing. Modify your inside-outside test to cast rays in multiple directions. Sucessfully voxelize a mesh which is not completely closed.
2. In the sample code we naively iterate over every triangle in the mesh when doing inside-outside testing. Implement an acceleration structure (Kd-Tree, Bounding Volume Hierarchy) to speed things up.

4 Submission Instructions

Please provide a report with your submission (PDF). The report should include the following:

- Images of voxelizations of all example files at 32x32x32 resolution and 64x64x64 resolution.

- Were there any references (books, papers, websites, etc.) that you found particularly helpful for completing your assignment? Please provide a list.
- Are there any known problems with your code? If so, please provide a list and, if possible, describe what you think the cause is and how you might fix them if you had more time or motivation. This is very important, as we're much more likely to assign partial credit if you help us understand what's going on.
- Did you do any of the extra credit? If so, let us know how to use the additional features. If there was a substantial amount of work involved, describe what how you did it. Provide at least one example for each extra feature implemented.
- Got any comments about this assignment that you'd like to share?

Remember, these assignments are to be done on your own. Please do not share code or implementation details with other students.

Submit your assignment on e-dimension by **May 30th by 11:55pm**. Please submit a single archive (`.zip` or `.tar.gz`) containing:

- Your source code (**which must be able to be built using CMake**).
- A compiled executable named `a1`.
- Any additional files that are necessary.
- The `PDF` file.