A world of unbounded dreams

A perforated lamp generating program

Downloads

Code: https://github.com/jiayinlu19960224/perf_lamp_final

Example models: https://www.thingiverse.com/thing:4053336

Video: https://youtu.be/5gCAXGaDBlM

If you have questions using the code, or find any errors of the documentation, or have any suggestions on improvements, please reach me at jiayinlu19960224@gmail.com. I am sorry for any potential errors in the document, as it was crammed out overnight.

This is my final project for a really fun course, Harvard GSD course SCI 6338, Introduction to Computational Design: https://www.gsd.harvard.edu/course/introduction-to-computational-design-fall-2019/, taught by Professor Jose Luis Garcia del Castillo Lopez: https://www.gsd.harvard.edu/person/jose-luis-garcia-del-castillo-lopez/.

Catalog

Introduction

I wrote a program in C++ to generate perforated lamps.

The program takes in two kinds of inputs: a lamp shape STL model, and pre-designed light projection patterns on the 6 surrounding walls of the lamp. The program will then generate holes at the corresponding places on the lamp model and output a perforated lamp STL model. The perforated lamp, when put in a light source inside, will then project the desired patterns on the walls.

The program is made available for anyone who wants to use it for their own design purposes on: https://github.com/jiayinlu19960224/perf_lamp_final .

I have also provided a *User Manual* section here, to aid with the use of the program.

The program is far from optimized, because of the time constraint I have for the project. It is currently just an unclean but working version. However, if people use the program, I will make some improvements on the code and the code structure and methods, to make it more efficient, and have a cleaner code structure.

For the purpose of demonstration, I made three lamp prototypes with the program. They all have the same desired wall projection patterns, but they are of different shapes. The goal is to demonstrate that the program can generate holes at the correct places of these different models, for the same wall patterns. They are shown in detail in the *Example* section.

Motivation

I have always liked light. Light is romantic; Light is imaginative; light is colorful; light is mysterious; light represents hope; light is a world of unbounded dreams.

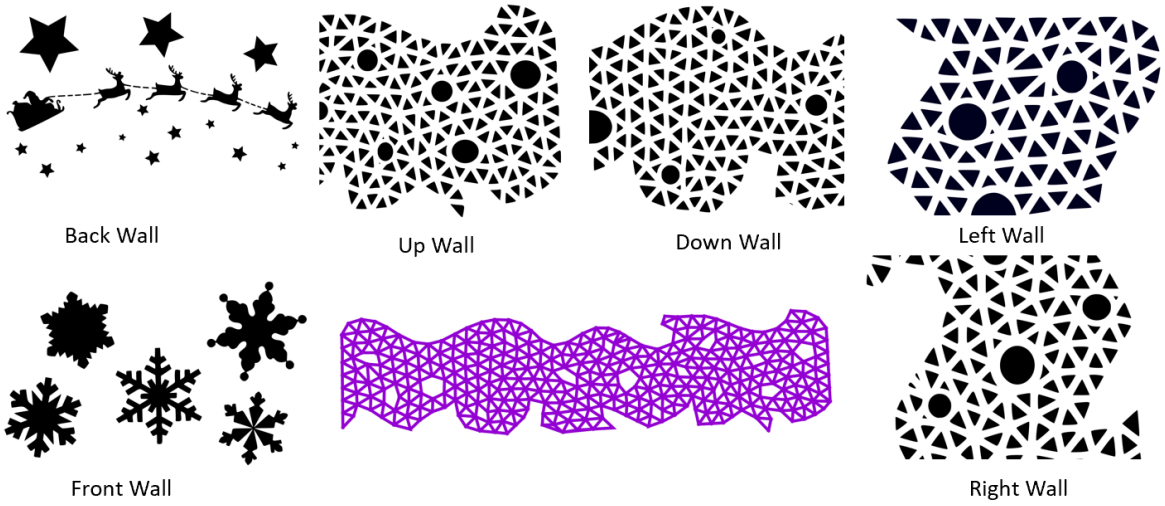
Earlier this semester, I read a post [1] by Jason Cole on stereographic lampshades. As seen in the figure below, he suggested a way to make perforated lamps with pre-designed light projection patterns. However, his method only works for spherical shape lamps, because he uses an explicit mathematical relationship matching the patterns to a sphere surface.

I thought the post is really cool, and I want to make my own perforated lamps as well. However, I want my lamps to be of any shapes, rather than just spheres. Also, I want to be able to define patterns for all the six walls surrounding the lamp.

Examples

I made three example prototypes, all with the same wall projection patterns, and all have different lamp shapes.

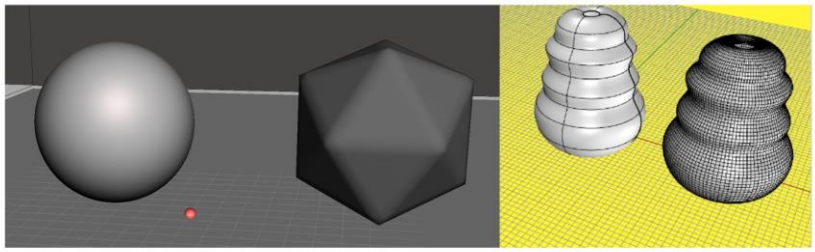
The wall projection patterns are defined as following:



Note that the “up”, “down”, “left” and “right” walls’ patterns would connect and form a pattern as shown in purple.

My three lamp base shapes are:

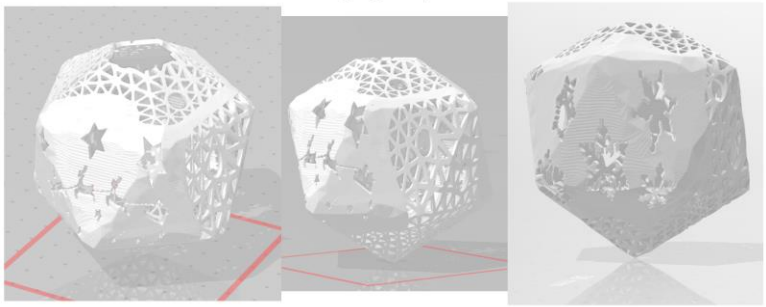
Primitives from Meshmixer software Shape using Rhino



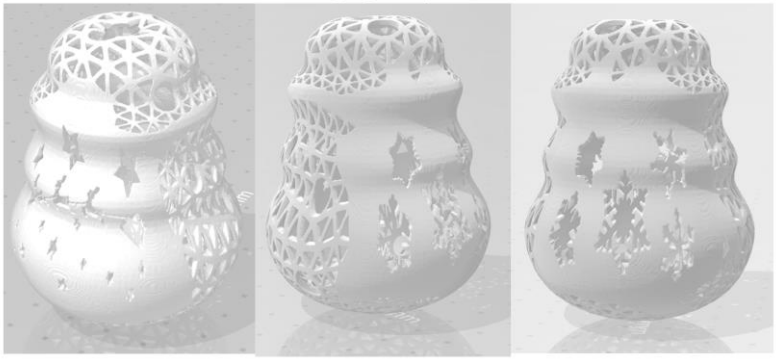
Note that to input into the program, we need to pre-process the above shapes. After processing, the input should be a shell of the lamp shape with a certain thickness, and a hole cut at the end where you want the lightbulb to go in.

With the above two inputs, I have the following outputs from the program:

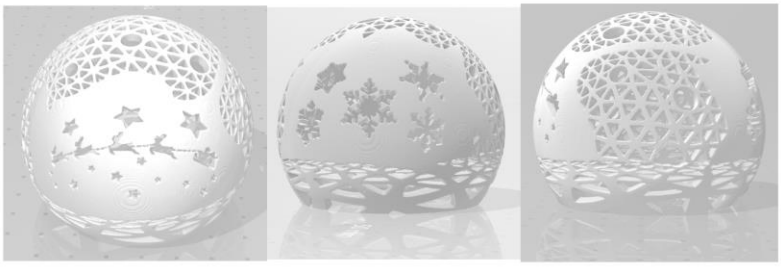
Hanging Lamp 1



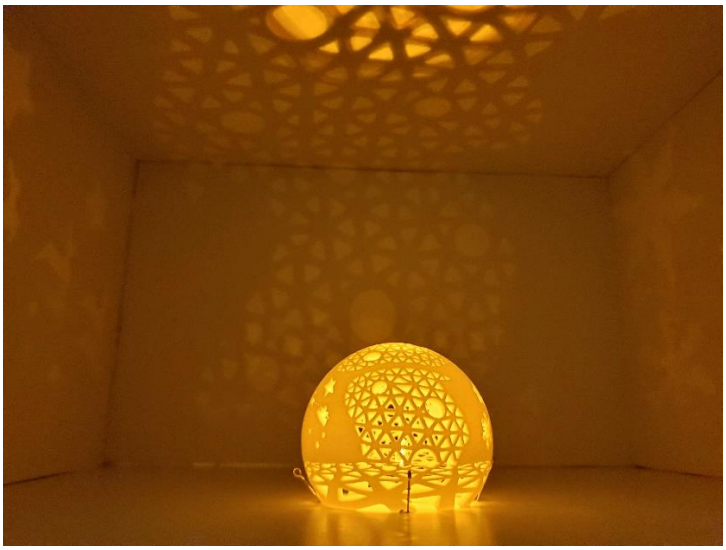
Hanging Lamp 2



Sitting Lamp 1



After setting up the lamps with colorful LED lightbulbs:



Sitting lamp: *Bedtime story*

Hanging lamp 1: *Goodbye to the past*

Hanging lamp 2: *Dancing into the night*

Note an imperfection in the physical setup is that, the LED lightbulbs does not generate much light in the bottom half. The light is only concentrated on the top of the lightbulb, therefore, the wall patterns corresponding to the bottom half of the LED lightbulbs are not projected well. The effect would be more ideal if I were able to obtain lightbulbs that emit lights in all directions.

I also made a much bigger version of one of the lamp models, so people can hold it in their hands and play with the light projections. I taped together different colors of LED lightbulbs (blue, green, white), which gives a dreamy, artistic lighting effect, as shown below.

Interactive holding lamp demo

Note that my example models are all available for free download on:
https://www.thingiverse.com/thing:4053336

User Manual

1. g++/gcc compiler

First of all, your computer will need to be able to run C++ programs with g++/gcc compiler. There are a lot of online resources on downloading and compiling C++ programs with g++/gcc.

2. Using the program

Suppose now your computer can run C++ programs, you can download the following from the Github repository:

- *perf_lamp.cc*: source code of the program

And then, in the same directory of where you put the *perf_lamp.cc* program, download an external dependency library [2]*eigen*.

Now, you can open *perf_lamp.cc* in text editors, or in IDEs like NetBeans or Visual Studio, and customize the program for your own design.

Please scroll down to *int main() {}:*

Here, you can input your lamp shape model in the *voxelizer()* function. In the example below, my lamp shape model is *shape3.stl*.

```
    //1. model
    //read in stl
    //scale the model
    //voxelize model
printf("A.\n");
    Voxelizer<double> voxelizer("shape3.stl", 0.002);  //0.002
printf("B.\n");
    voxelizer.AdvancedVoxelization();
    double dx_half=0.5*voxelizer._dx;
```

Then, you can input your six walls' projection patterns.

The projection patterns should be from binary (black and white) images and stored at .txt files.

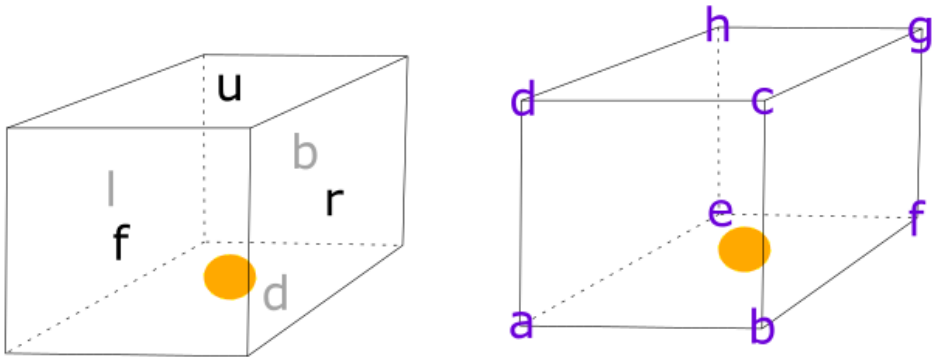
For example, for an image of 600(rows)x1000(columns) pixels, all the light projected pixels should be labeled "1" in the .txt file, and all the un-projected pixels should be labeled "0" in the .txt file.

For the following, you will need to type in the corresponding pixels (m for number of rows, n for number of columns) for the six wall projections: b is back wall, f is front wall, r is right wall, l is left wall, u is up wall, d is down wall. You will also type in the corresponding .txt files for the projection images and their pixel values inside the *read_image()* function.

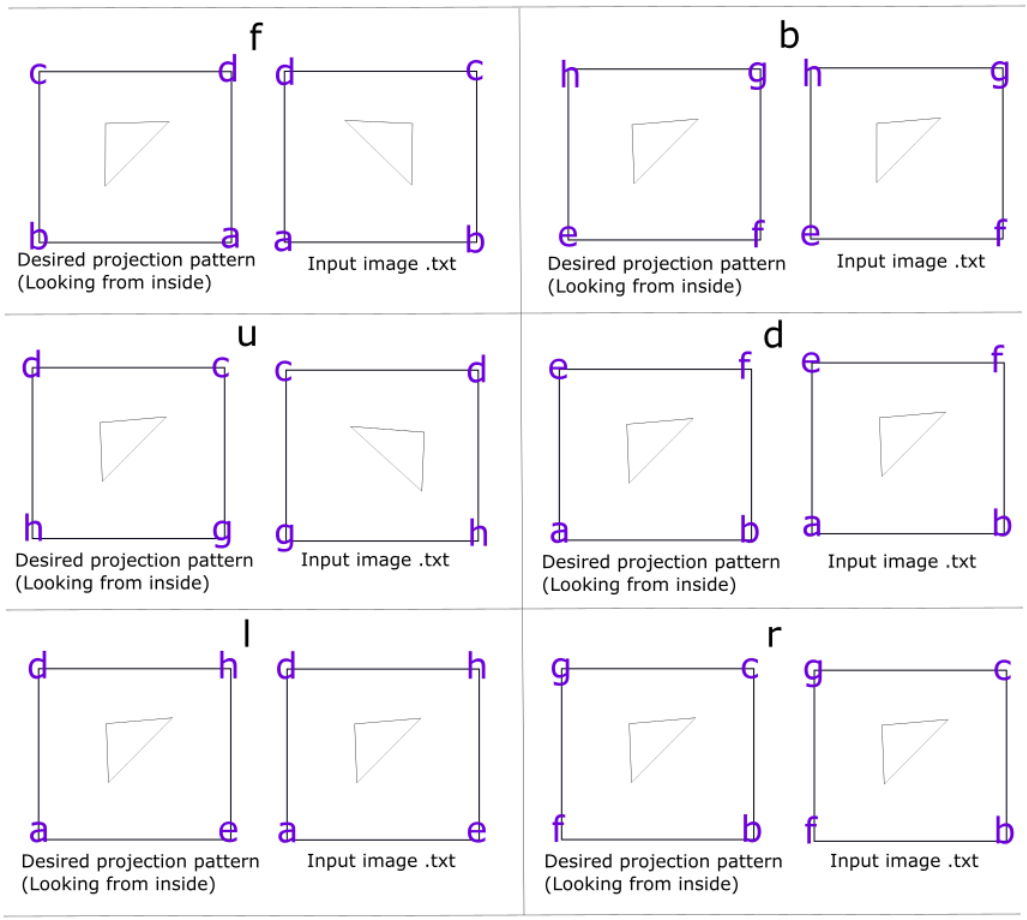
```
//2. patterns
//read in projection images binary pixel patterns in .txt (6 surrounding walls)
int b_pixel_m=600; int b_pixel_n=1000;
int f_pixel_m=600; int f_pixel_n=1000;
int r_pixel_m=600; int r_pixel_n=1000;
int l_pixel_m=600; int l_pixel_n=1000;
int u_pixel_m=1000; int u_pixel_n=1000;
int d_pixel_m=2000; int d_pixel_n=2000;

read_image("santa_deer.txt", b_pixel_m, b_pixel_n, &bgrid);
read_image("snow_flakes.txt", f_pixel_m, f_pixel_n, &fgrid);
read_image("r_Tria_pattern.txt", r_pixel_m, r_pixel_n, &rgrid);
read_image("l_Tria_pattern.txt", l_pixel_m, l_pixel_n, &lgrid);
read_image("u_Tria_pattern.txt", u_pixel_m, u_pixel_n, &ugrid);
read_image("d_Tria_pattern.txt", d_pixel_m, d_pixel_n, &dgrid);
```

Note that I use the following orientation for the walls input. Suppose we have a space like this, with the walls and wall corners labeled:



Then the projected patterns .txt input should have the following orientations:



Next, we define how far away the walls are from the lamp.

The program automatically shrinks and centers the model when reading in the STL file. It shrinks and centers the model to be inside a bounding box of x: [-0.5, 0.5], y: [-0.5, 0.5] and z: [-0.5, 0.5]. The model is shrunk such that the maximum dimension of the model is of length 1.

Further, the model is translated so that its bottom (minimum z) always sit on z=-0.5.

Now, knowing where the model locates, we can set the surrounding walls. The surrounding rectangular walls would be determined by six parameters: xmin, xmax, ymin, ymax, zmin and zmax. You can set your wall locations related to the shrunk model below.

```
//define surrounding walls domain
double wall_xmin=-2.5; //bgrid
double wall_xmax=2.5;  //fgrid
double wall_xrange=wall_xmax-wall_xmin;
double wall_ymin=-2.5; //lgrid
double wall_ymax=2.5;  //rgrid
double wall_yrange=wall_ymax-wall_ymin;
double wall_zmin=-2.5; //dgrid: shape2
//double wall_zmin=-0.5; //dgrid: spherical
double wall_zmax=2.5;  //ugrid
double wall_zrange=wall_zmax-wall_zmin;
```

Lastly, we define the location of the light source (lsx, lsy, lsz) related to the shrunk lamp in the first three lines of code below:

```
//3. putting together
//define a light source location
double lsx=0.0;
double lsy=0.0;
double lsz=voxelizer.model_zmin+0.5*(voxelizer.model_zmax-voxelizer.model_zmin); //at zmid: lamp shape 2
//double lsz=voxelizer.model_zmin+1.0/3.0*(voxelizer.model_zmax-voxelizer.model_zmin); //at 1/3 height: spherical
//voxel grid (lsi, lsj, lsk) that the light source is in.
int lsi=(int)((lsx-voxelizer._pmin[0])/voxelizer._dx);
int lsj=(int)((lsy-voxelizer._pmin[1])/voxelizer._dx);
int lsk=(int)((lsz-voxelizer._pmin[2])/voxelizer._dx);
```

3. Running the program

Now you can compile and run the program with the following command in a Unix shell:

```
g++ -I ~/Desktop/Kay/projects/perforatedLamp/External/eigen -o perf_lamp perf_lamp.cc -fopenmp
```

Please replace the path above to your own path to the external dependency *eigen* folder.

Then you will obtain a perforated lamp STL model back in the local repository.

The running should take only a couple minutes.

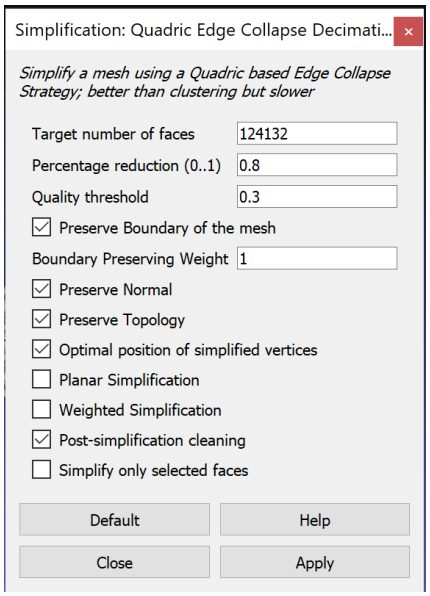
4. Post-processing the output model

However, the STL model you obtained is very large, because it was generated from a voxel model format. I found the following cleaning up procedure very helpful:

- First, use MeshMixer: Edit -> Make Solid -> choose “accurate” and a high solid accuracy (~470) and a high mesh density (~400). Save the updated model.

  You may also want to resize the model here to your desired size, as the output model is very tiny (1mm length in the maximum model dimension).

- Then, import the updated model into MeshLab, in order to simplify the mesh and reduce the file size: Filters -> Remeshing, Simplification and Reconstruction -> Simplification: Quadric Edge Collapse Decimation: Go through a couple times of the operations with the following settings, until you reach a desired file size.



Now, you can save the model and print it.

Implementation

I use a computer graphics approach on the problem. Here is an outline of the steps.

1. Read and store the triangle meshes of the model from STL file (Code from [3]);
2. Shrinks and center the model;
3. Voxelize the model;
4. Define the light source and wall locations;
5. Link each pattern geometry points in the image with the light source point, as a “ray”, and do Boolean operation to void the voxel that the ray goes through;
6. Marching cube algorithm to convert voxels back to triangular mesh (Code from [3]);
7. Write the new triangular mesh to output STL file (Code from [3]).

Reference

[1] Jason Cole, *Spherical stereographic lamp*, https://jasmcole.com/2014/11/01/stereographic-lampshades/

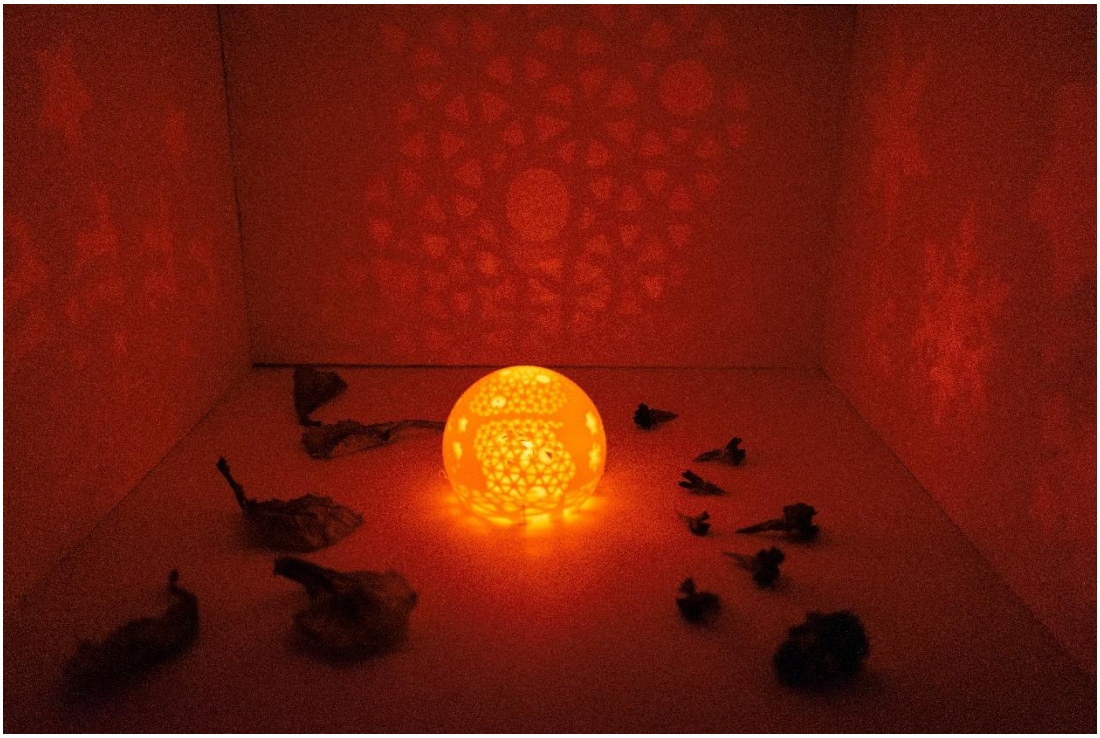
[2] eigen, *a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms*, http://eigen.tuxfamily.org/index.php?title=Main_Page

[3] homework code from course *MIT 6.839 Advanced Computer Graphics*

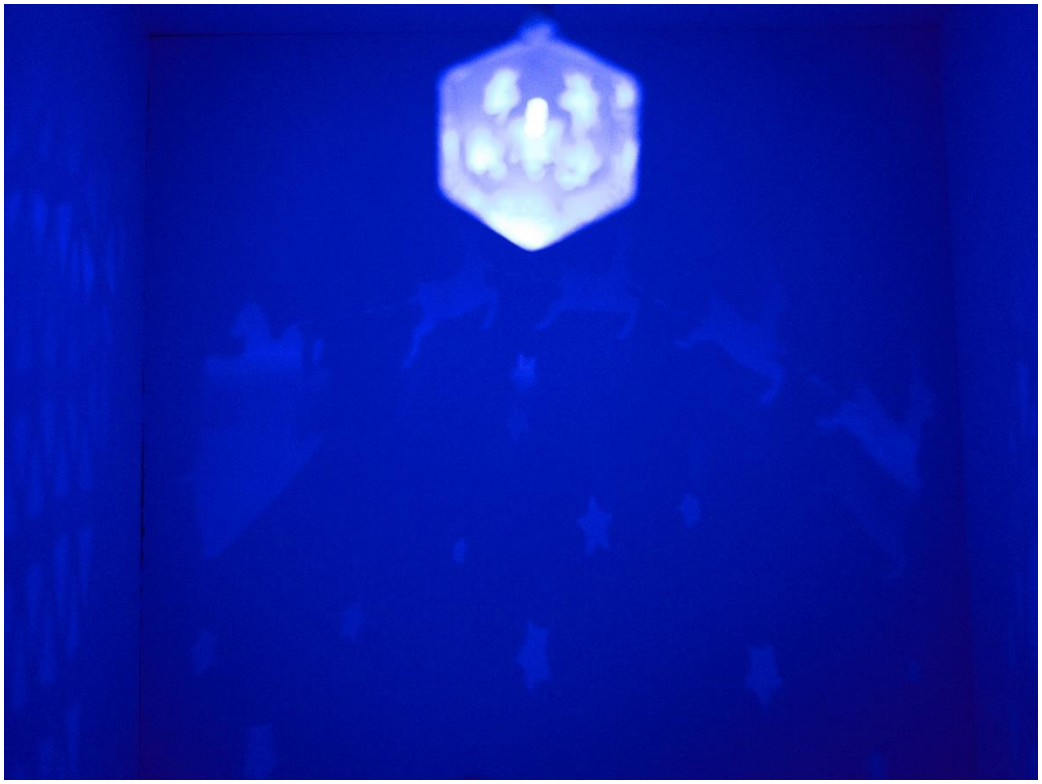
Artwork Gallery

My three example pieces, each of them is an artwork with a theme.

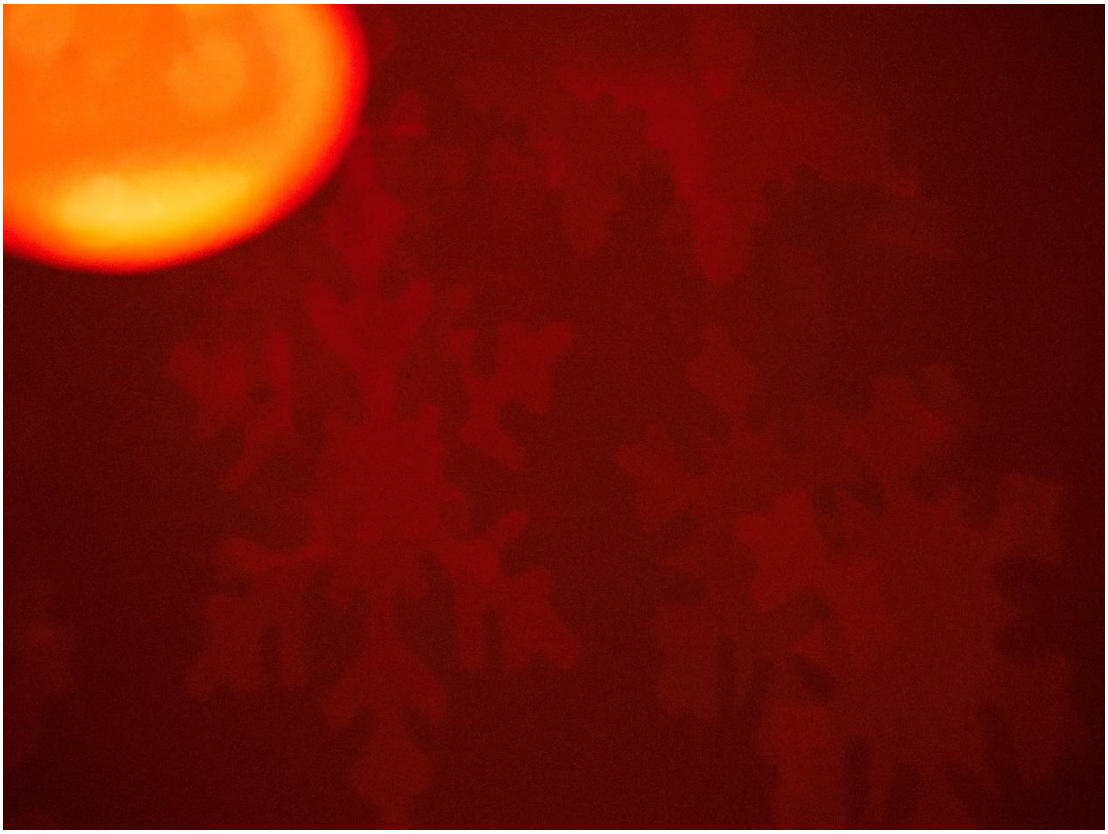
Bedtime Story

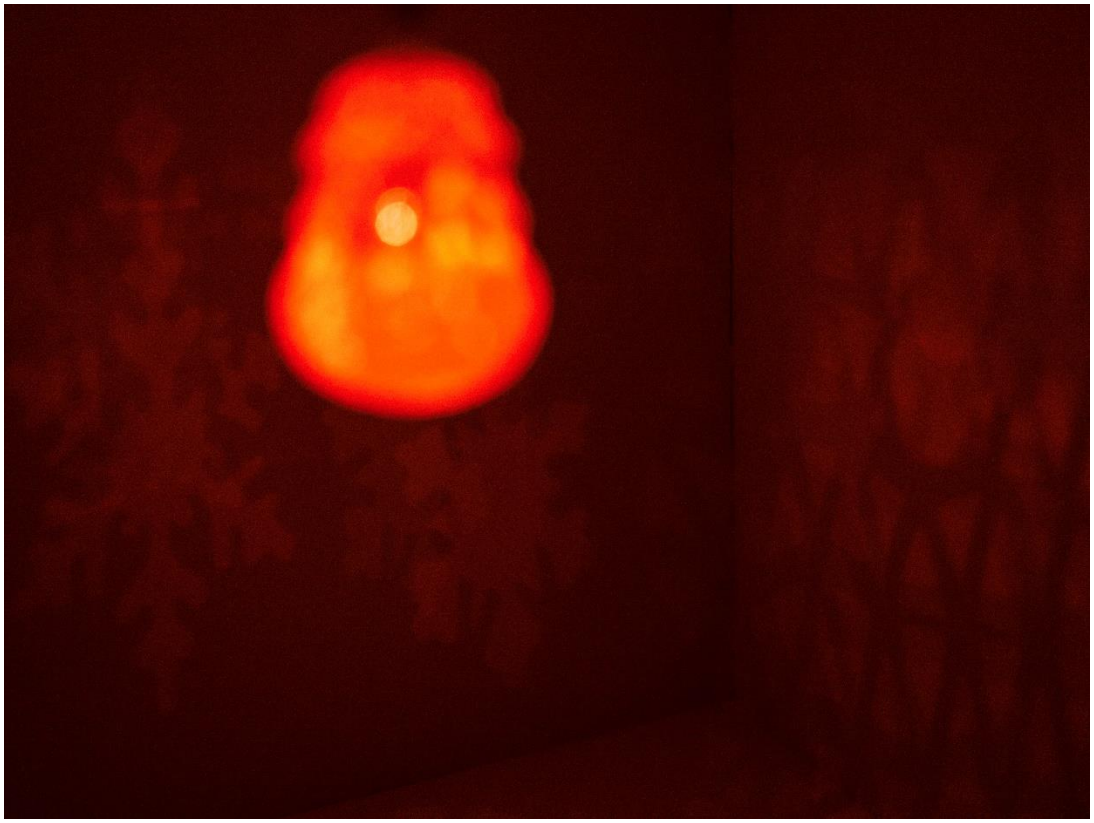


Goodbye to the past



Dancing into the night





Future Improvements:

There are a lot of improvements possible in the code implementation:

1. Make it into a standard class, with .cc, .hh, and Make file.
2. Currently, in Implementation Step 5, I am linking line segments between the light source point and each lighted pattern pixel point, and sample x (or y, or z) values along the line segment, to void the voxel that this (x,y,z) point lies in. This implementation is good that it avoids looping through all the voxels, therefore saving a lot of time.
   However, I think there are better ways to implement this as well. One way is to only consider the pixels at the edge of the pattern, and then maybe do a polygon-cube intersection detection. Or other ways, like knowing the inside and outside of the shape to cut out, and then go through z-axis of the voxel model layer by layer to cut out the parts inside the polygon.
3. Currently, the code write out the voxel info and then read it in again to run the marching cube algorithm. The writing out voxel info step can be eliminated. This is an inefficiency in the code that can be easily fixed later.

I was also actually hoping to make more lamps with more interesting shapes and patterns. I was thinking a lot about the shapes after visiting the Philadelphia Museum of Art. I think a lot of inspiration could be drawn from their exhibition (See photos below). But because of the time constraint I didn't make more designs. Hopefully other people can use the program to make more interesting designs!