CS171 HW0 Recitation

10/4/2019

Homework is available on the course website!

HW0 here:

http://courses.cms.caltech.edu/cs171/assignments/hw0/hw0-html/cs171hw0.html

Notes on Geometric Transformations:

http://courses.cms.caltech.edu/cs171/assignments/hw1/hw1-notes/notes-hw1.html

Class Virtual Machine:

http://courses.cms.caltech.edu/cs171/materials/171_vm.ova (will be updated soon)

Fill out the office hours survey! Closing 7:30pm tonight.

Primer on geometric transformations

- Given some geometry, we also want a way to customize it.
    - *“I want a larger bunny…”*
- Three main transformations are **translation, rotation,** and **scaling.**
- Transformations can conveniently be represented by matrices.

$$\mathbf{Mx} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

- In practice, we use 4 x 4 matrices.

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- Homogeneous coordinates allow translation and perspective projection to be expressed as matrices.

Homogeneous coordinates

- For a given point p in 3D space with Cartesian coordinates *(X, Y, Z)*, we express it in homogeneous coordinates *(x, y, z, w)* where *w* is known as the homogeneous component.

$$(X, Y, Z) \rightarrow (x, y, z, w) \begin{cases} X = \frac{x}{w} \\ Y = \frac{y}{w} \\ Z = \frac{z}{w} \end{cases}$$

Translation Matrix

$$\begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + v_x \\ y + v_y \\ z + v_z \\ 1 \end{bmatrix}$$

Rotation Matrix

- Rotating about an axis in the direction of the unit vector u counterclockwise by an angle θ
  - Make sure your vector is a unit vector -- check for this.

$$R = \begin{bmatrix} u_x^2 + \left(1 - u_x^2\right)\cos\theta & u_x u_y \left(1 - \cos\theta\right) - u_z \sin\theta & u_x u_z \left(1 - \cos\theta\right) + u_y \sin\theta & 0 \\ u_y u_x \left(1 - \cos\theta\right) + u_z \sin\theta & u_y^2 + \left(1 - u_y^2\right)\cos\theta & u_y u_z \left(1 - \cos\theta\right) - u_x \sin\theta & 0 \\ u_z u_x \left(1 - \cos\theta\right) - u_y \sin\theta & u_z u_y \left(1 - \cos\theta\right) + u_x \sin\theta & u_z^2 + \left(1 - u_z^2\right)\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How to check if your rotation matrix is correct

- Rotation matrices are orthogonal
- Easy way to check is by multiplying your rotation matrix by its transpose
    - Should get the identity matrix out
    - $R * R^T = I = R^T * R$

Scaling Matrices

$$S = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Combining transformations

We can multiply all transformation matrices together into a single matrix.

Order is important!

- Matrix multiplication is not necessarily commutative.
- For example, If we tell you to Scale the points (S), translate the points (T), Rotate the points (R), and then translate again (T2), with the point vector v.
  - The sequence of matrix multiplications should look like: (T2)(R)(T)(S)(v)

Purpose of this week’s assignment

- Write functions to parse files containing geometry and their transformations.
- Practice outputting a simple image.
- Get set up for the rest of the term’s assignments. You will use these functions over and over!

Part 0: Setting up OpenGL

- Nothing to turn in!
- Set up OpenGL on your computer, which will be used in a couple weeks.
- We provide a demo program for you to run, to make sure everything is working properly.
- Installation guide available on website - ask TAs if you run into any difficulties.

TAs: *Alden Rogers, Nicole Feng, Ethan Jaszewski*

Resources: *Piazza, office hours*

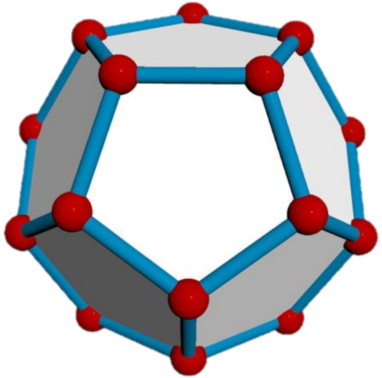
Part 1: Parsing .obj files

What are .obj files?

Files that describe polygonal geometry.

What information is needed to specify such geometry?

At minimum, *vertices* and *faces*.



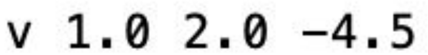
wordpress.discretization.de

Part 1: Parsing .obj files

How are vertices specified?

Vertices are specified by a ‘v’, followed by 3 space-separated floats corresponding to XYZ Cartesian coordinates.

Example:



```
v 1.0 2.0 -4.5
```

indicates a vertex at position (1.0, 2.0, -4.5).

Part 1: Parsing .obj files

How are faces specified?

Faces are specified with an ‘f’ followed by 3 space-separated ints corresponding to the indices of its constituent vertices.

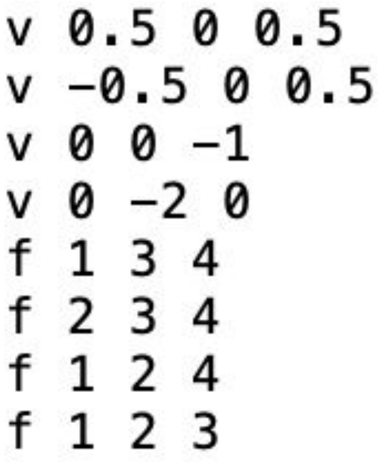
Example:

```
f 1 3 4
```

This tells you how to connect 3 points to make an oriented triangle.

****Note that vertices are 1-indexed, NOT 0-indexed!*

An example .obj file



```
v 0.5 0 0.5
v -0.5 0 0.5
v 0 0 -1
v 0 -2 0
f 1 3 4
f 2 3 4
f 1 2 4
f 1 2 3
```

Each line specifies either a vertex or a face.
The vertex list is followed by the face list.

Part 1: Converting .obj files to usable data structures

- We need to convert the .obj file contents into objects we can operate on.
- There is no “correct” way to do this, except we recommend that you make structs that are **simple** and **extendable,** since they will be re-used and built upon in the following weeks’ assignments.
- *Possibilities:* Since both vertices and faces have 3 fields, you can have a struct for vertices and a struct for faces, or one that works for both. Then have a vector that stores all the vertices, and a vector for all the faces.

Part 1: Command line arguments

You are getting the file names of the obj files to parse from the command line.

```
int main(int argc, char *argv [] ) {
        …
        // stuff
        …
}
```

argc = 1 + (# of arguments)

argv: arguments start with the second entry of the array

(1st element is just the name of the program)

Part 1: File I/O

- I recommend including **fstream** (feel free to do I/O a different way)
- Use **std::ifstream infile(filename)** to open the file stream
- Then you can use a while loop
  - While (infile >> x)
  - Puts the next token (whitespace separated by default) into x
  - Can extend this to have multiple token grabs in the while conditional
  - Ex: while (infile >> a >> b >> c >> d)
  - sstream is another possibility

Part 1: Printing out the .obj files you read in

- You’re just going to read in .obj files and spit them back out to see if you processed them correctly.
- Print the files out in order that they were given.
- You can simply use **cout** or **printf.**

Part 2: Working with Eigen

- Eigen is a linear algebra library we will use in this course.
- It will be included in the zip file for hw0, just tell your makefile to look up one directory after the -I, i.e. `-I ../`
- Might have some deprecation warnings, ignore them for now.

  (Or add `-Wno-deprecated-declarations`)

Eigen Matrices

```
Matrix4d m;

m << 4, 11, 7, 2,       // row1

     0, 5, 6, 7,        // row2

      1, 15, 12, 7,    // row3

      13, 0, 12, 10; // row4
```

```
Matrix4d m;

m << 4, 11, 7, 2, 0, 5, 6, 7, 1, 15, 12, 7, 13,
0, 12, 10;
```

Eigen cheat sheet: http://eigen.tuxfamily.org/dox/AsciiQuickReference.txt

Part 2: Parsing the input file

- Part 2 involves reading in vectors encoding various transformations.
- You can just adapt your Part 1 code.

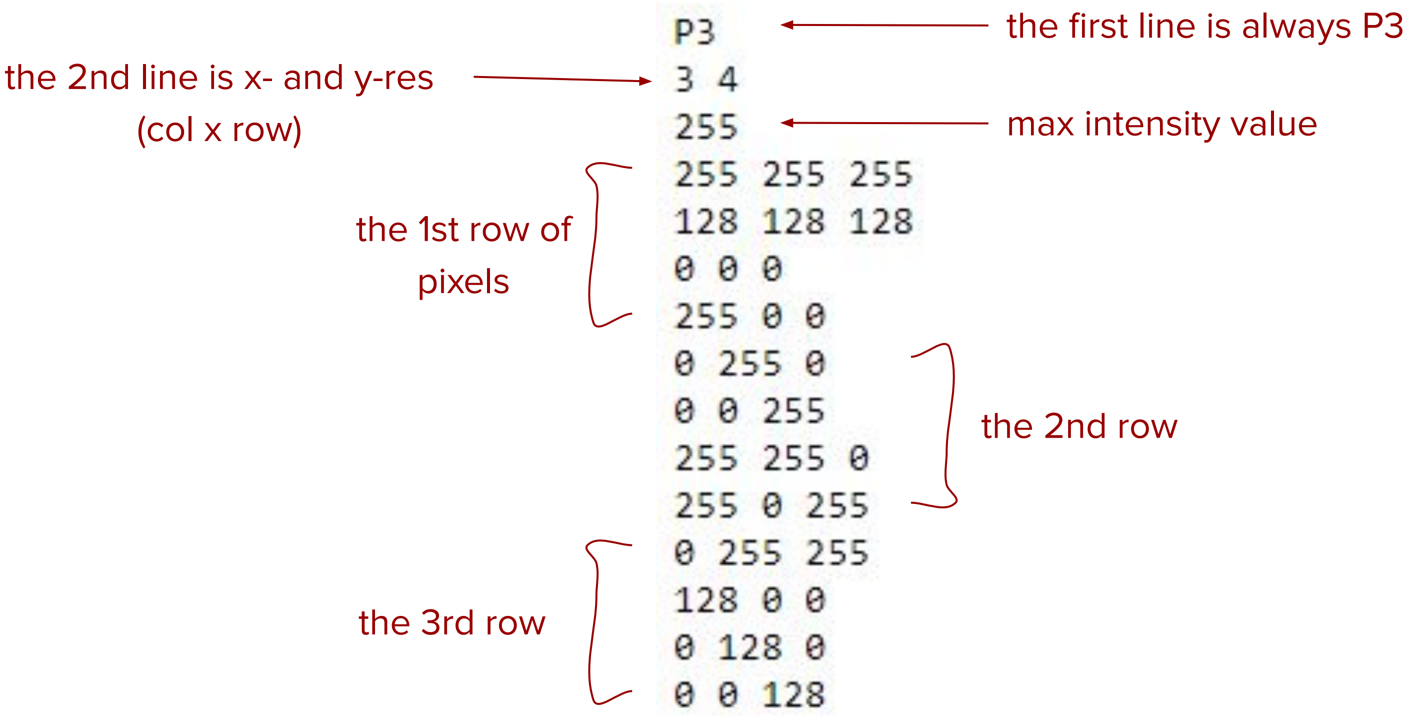
Part 3: Putting the two programs together

- Similar parsing as in Part 1
- Be careful, you might load the same object twice but have different transformations for it
    - Be sure to name them properly to tell them apart.

Part 4: The PPM Image Format

- The first line is always P3
- The second line specifies x- and y- resolutions (space separated)
- Third line is the maximum pixel intensity (we used 255 in example)
- Each subsequent line should have 3 numbers between 0 and max intensity
  - RGB values for pixels in grid from left to right, top to bottom
- Just print out to terminal
- Assignment gives example how to view the image you create

PPM example



the first line is always P3

the 2nd line is x- and y-res (col x row)

max intensity value

```
P3
3 4
255
255 255 255
128 128 128
0 0 0
255 0 0
0 255 0
0 0 255
255 255 0
255 0 255
0 255 255
128 0 0
0 128 0
0 0 128
```

the 1st row of pixels

the 2nd row

the 3rd row

Reminders

- HW due next week Wednesday at 3pm
- Office Hours TBD
- You will use your code from parts 1, 2, and 3 again next week!
- If you do a nice job, organize your code nicely into classes/functions you'll save time in the future
- Please use functions. Don't just stick everything in main...
- Submit to Moodle