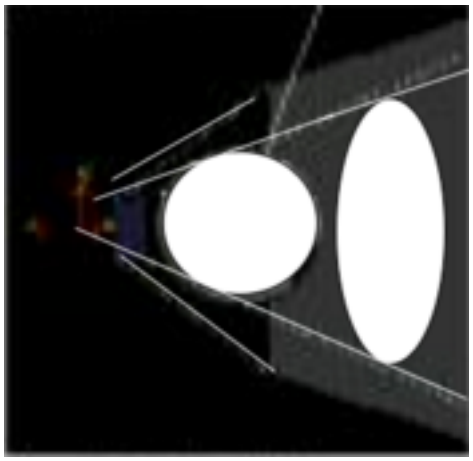
CMSC 634/435 Introduction to Computer GProject

Assignment 2: Viewing: What do you see…

(Due March 9th, 11:59pm)



Goals of this project: Understand viewing matrix setup in computer graphics and OpenGL. Once this project is completed, you will be able to display 3D objects oriented in any way and viewed from any position. You will also learn OpenGL texture setup.

Introduction:

So far in project 1, we have learned how to construct a 3D scene from geometry primitives and their transforms. This programming assignment will let us learn how to start fro a point in 3D to compute its projection into the image on the 2D screen. Basically, how exactly the pixel is drawn on the screen. The central tool is again matrix transformations.

What you will learn is that the seemingly complicated viewing concept can be decomposed into many simpler concepts, each of which can be represented by a transformation matrix. What we have learned in class is to combine matrices seamlessly with coordinate transformations used to position camera and geometry in the scene (Camera transform); we also learned projection transform (from Camera to canonical view) and windowing transformation (from canonical to screen). Ultimately, we only need to multiply this matrix to map any 3D point to its correct screen location.

In this assignment, we will exercise the placement of a sphere on the screen. There are two types of projections we want to implement in this assignment.

Orthographic transformation chain:

- Start with co-ordinates in object's local coordinates
- Transform into world co-ordinates(Modeling transform, M_m)
- Transform into eye co-ordinates, M_{cam} (Camera co-ordinates)
- Orthographic projection, M_{orth}
- Viewport transform, M_{vp}

$$\mathbf{P_s = M_{vp}M_{orth}M_{cam}M_mP_o}$$

Perspective transformation chain (634 only):

- Transform into world co-ordinates(modeling transform, M_m)
- Transform into eye co-ordinates $M_{cam.}$
- Perspective matrix, P
- Orthographic projection, $M_{orth.}$
- Viewport transform, $M_{vp.}$

$$\mathbf{P_s = M_{vp}M_{orth}PM_{cam}M_mP_o}$$

Requirement:

Your viewing program should support one geometry type:

- Sphere

The output should be a plane in the 3D scene and the viewed picture should be displayed on the plane. The result is similar to the OpenGL example code Dr. Chen showed in class. Please note that you MUST use OpenGL to compute the image yourself. Once you obtain the projected image on the screen, use it as a texture and attach it to the OpenGL plane.

For students in 435, you only need to implement the orthogonal projection condition.

For students in 634, you must implement perspective projection.

Choose your own plane size and place it at the center of the coordinate system passing (0, 0, 0);

Input file format:

- *Surface*: This element describes a geometry object. It must have an attribute *type* with value *Sphere* (we won't use any other geometries). For Sphere, *center* containing a 3D point, and *radius* containing a real number.

- *Camera*: This element describes the camera. It is described by the following elements:
    - *viewPoint*, a 3D point that specifies the center of projection.
    - *viewDir*, a 3D vector that specifies the direction toward which the camera is looking. Its magnitude is not used.
    - *viewUp*, a 3D vector that is used to determine the orientation of the iamge.
    - *projNormal*, a 3D vector that specifies the normal to the projection plane. Its magnitude is not used, and negating its direction has no effect. By default it is equal to the view direction.
    - *projDistance*, a real number *d* giving the distance from the center of the image rectangle to the center of the projection.
    - *viewWidth and viewHeight,* two real numbers that give the dimensions of viewing window on the image plane.

  The camera's view is determined by the center of projection (the viewpoint) and a view window of size *viewWidth* and *viewHeight*. The window's center is positioned along the view direction at a distance d from the viewpoint. It is oriented in space so that it is perpendicular to the image plane normal and its top and bottom edges are perpendicular to the up vector.

- *Image:* This element is just a pair of integers that specify the size of the output image in pixels.

Your code only needs to handle the following scene which contains a single sphere or cube.

For Sphere:

viewpoint : 5 4 3

viewDir: -5 -4 -3

projNormal: 5 4 3

viewUp: 0 1 0

projDistance: 5

viewWidth: 2.5

viewHeight: 2.5

image size: 64 64

sphereCenter: 0 0 0

sphereRadius: 1

For 634 (graduate students only):

You will need to implement the perspective projection.

Supporting code:

Dr. Chen has added texture display function to the code used for project 1. She uses a checkboard display to show you how to do it. Please change it to the image computed through viewing transform.

Algorithm hint:

Undergraduate only needs to compute projection points from a bounding circle. Once your code computes the locations of these pixels on the viewing plane (defined by viewWidth and viewHeight), use the scanline algorithm to fill in all internal pixels using the same color as the bounding circle.

What to turn in

Source code only by email to Grader. Please do not include any .o files. Please include:

- A README with your handing containing basic information about your design decisions and any known bugs or extra credit;
- How to compile and run your code as if you are telling a colleague that is to continue the development. This means you will need to submit **all code**.
- Please name your project directory as **02viewing_<your umbc name>** and put everything in that directory.

You only need to submit ONE tarball that compresses all files in the ***02viewing_<your umbc name>*** directory. To create the tarball, on the gl server, go one level above the 02viewing_<your umbc name> directory and then type in the following line.

tar cfv 02viewing_<your umbc name>.tar 02viewing_<your umbc name>

Email 02viewing_<your umbc name>.tar to the Grader Kevin.

Project 2 Rubric:

Points Possible	Description
25 (435)/10 (634)	Code compiles
5	Code is well documented and contains a README file

10(435)/10(634)	Define sphere geometry
45 (435) / 35 (634)	Apply orthographic projection matrix transformations
10 (435 extra credit, 634)	Apply Perspective projection matrix transformation
15 (435, 635)	Draw the texture plane and attach the image
15 (435 extra credit, 634)	Draw the viewing frustum and the camera in your 3D scene

Total: 100. If code does not compile on gl, Score will be 0.