Programming Assignment #4

CS 200, Fall 2024

Due Friday, October 4

I will give you the header file `Camera.h`, which contains the following declarations.

```
namespace cs200 {
  class Camera {
    public:
      Camera(void);
      Camera(const glm::vec4 &C, const glm::vec4 &v, float W, float H);
      glm::vec4 center(void) const             { return center_point; }
      glm::vec4 right(void) const              { return right_vector; }
      glm::vec4 up(void) const                 { return up_vector; }
      float width(void) const                  { return rect_width; }
      float height(void) const                 { return rect_height; }
      Camera& moveRight(float x);
      Camera& moveUp(float y);
      Camera& rotate(float t);
      Camera& zoom(float f);
    private:
      glm::vec4 center_point;
      glm::vec4 right_vector, up_vector;
      float rect_width, rect_height;
  };

  glm::mat4 affineInverse(const glm::mat4 &A);
  glm::mat4 cameraToWorld(const Camera& cam);
  glm::mat4 worldToCamera(const Camera& cam);
  glm::mat4 cameraToNDC(const Camera& cam);
  glm::mat4 NDCToCamera(const Camera& cam);
}
```

(the `Affine.h` header file has been included). You are to implement all of the declared functions, which are described in detail below.

`Camera()` — (default constructor) creates a camera object whose world space viewport is the standard square: the square centered at the origin with width and height of 2.

`Camera(C,vp,W,H)` — (nondefault constructor) creates a camera object whose world space viewport is the rectangle centered at C, with right vector $\vec{u}$, up vector $\vec{v}$, width W, and height H. The (unit) vector $\vec{v}$ points in the same direction as the vector `vp`, and is a 90° counterclockwise rotation of the (unit) vector $\vec{u}$. Note that `vp` is *not* assumed to be a unit vector.

`center()` — returns the center of the camera in world coordinates. [Implemented]

`right()` — returns the (unit) right vector of the camera in world coordinates. [Implemented]

`up()` — returns the (unit) up vector of the camera in world coordinates. [Implemented]

`width()` — returns the width of the camera viewport in world coordinates. [Implemented]

`height()` — returns the height of the camera viewport in world coordinates. [Implemented]

`moveRight(d)` — moves the camera d world space units in a direction parallel the the camera's right vector. A reference to the camera is returned.

`moveUp(d)` — moves the camera d world space units in a direction parallel the the camera's up vector. A reference to the camera is returned.

`rotate(t)` — rotates the camera t degrees counterclockwise about the camera's center. A reference to the camera is returned.

`zoom(f)` — scales the dimensions of the camera viewport rectangle by a factor of f with respect the the camera's center. Note that the camera's aspect ratio is preserved after the call to `Zoom`. A reference to the camera is returned.

`affineInverse(A)` — returns the inverse of the 2D affine transformation A. It is the responsibility of the function caller to ensure that A is an invertible affine 2D transformation matrix. You do not need to check for this. However, you are expected to implement the method discussed in class.

`cameraToWorld(cam)` — returns the camera modeling transformation: the transformation that maps the camera space coordinate system of `cam` to the world space coordinate system.

`worldToCamera(cam)` — returns the viewing transformation: the transformation that maps the world space coordinate system to the camera space coordinate system of `cam`.

`cameraToNDC(cam)` — returns the device transformation: the transformation that maps the camera space coordinate system of `cam` to normalized device coordinates based on the standard square. This transformation maps the viewport in camera space (the axis aligned rectangle centered at the origin with the same dimensions as the world space viewport), to the square centered at the origin with width and height of 2.

`NDCToCamera(cam)` — returns the transformation that maps the normalized device coordinates based on the standard square to the camera space coordinate system of `cam`. This transformation maps the standard square to the camera space viewport rectangle.

What to turn in

Your submission for this assignment will consist of a single source file, which must be named `Camera.cpp`. You may include only the `Affine.h` and `Camera.h` header files.