Programming Assignment #3

CS 200, Fall 2015

Due Tuesday, September 22

Task #1: inverse of an affine transformation

In the header file `Affine.h` from the previous assignment, the function `Inverse` is declared as

```
  Affine Inverse(const Affine& A);
```

However, you did not implement this function. For this portion of the assignment, you will implement the `Inverse` function, which returns the inverse of affine transformation A. Although A is represented by a 3×3 matrix, you may assume that this matrix is of the form

$$A = \begin{bmatrix} L_{xx} & L_{xy} & v_x \\ L_{yx} & L_{yy} & v_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $\left(\begin{smallmatrix} L_{xx} & L_{xy} \\ L_{yx} & L_{yy} \end{smallmatrix}\right)$ is the matrix for the linear part of A and $\vec{v} = \langle v_x, v_y \rangle$ is the translation part; that is, $A = T_{\vec{v}} \circ L$. Therefore, the inverse of A is given by

$$A^{-1} = (T_{\vec{v}} \circ L)^{-1} = L^{-1} \circ T_{-\vec{v}}.$$

Since L is a 2×2 matrix, its inverse is given by the matrix

$$L^{-1} = \frac{1}{L_{xx}L_{yy} - L_{xy}L_{yx}} \begin{pmatrix} L_{yy} & -L_{xy} \\ -L_{yx} & L_{xx} \end{pmatrix}$$

— provided that $\det(L) = L_{xx}L_{yy} - L_{xy}L_{yx} \neq 0$. You do not have to check if the determinant of L is zero or not: this is the responsibility of the caller of the `Inverse` function.

Your submission for this part of the assignment should consist of a single file, named `Inverse.cpp`, which contains the implementation of the `Inverse` function only. You may only include the `Affine.h` header file, but you are free to use any of the functionality declared there.

Task #2: camera class

The header file `Camera.h` contains the following declarations.

```
class Camera {
  public:
    Camera(void);
    Camera(const Point& C, const Vector& up, float W, float H);
    Point Center(void) const;
    Vector Right(void) const;
    Vector Up(void) const;
    float Width(void) const;
    float Height(void) const;
    Camera& MoveRight(float x);
    Camera& MoveUp(float y);
    Camera& Rotate(float t);
    Camera& Zoom(float f);
  private:
    Point center;
    Vector right, up;
    float width, height;
};


Affine CameraToWorld(const Camera& cam);
Affine WorldToCamera(const Camera& cam);
Affine CameraToNDC(const Camera& cam);
Affine 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 whose world space viewport is the standard square: the square centered at the origin with width and height of 2.

`Camera(C,up,W,H)` — (nondefault constructor) creates a camera 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 `up`, and is a $90°$ counterclockwise rotation of the (unit) vector $\vec{u}$.

`Center()` — returns the center of the camera in world coordinates.

`Right()` — returns the right vector of the camera in world coordinates.

`Up()` — returns the up vector of the camera in world coordinates.

`Width()` — returns the width of the camera viewport in world coordinates.

`Height()` — returns the height of the camera viewport in world coordinates.

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

`MoveUp(y)` — moves the camera y 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 radians 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.

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

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

`CameraToNDC(cam)` — returns 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.

Your submission for this part of the 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.