Project 2 Part I

1.1 Overview

You should design basic data structures and classes for computer graphics.

- Part I: linear algebra (quaternions and transformations)
- Part II: integration of part I into an OpenGL framework

1.2 Required Functionality: Part I

Here are the elements that your program must have.

- **Quaternions:** Implement quaternions and their math in file quat.h.

- **Transformation matrices:** extend mat.h by the following methods and implement them.

```
static mat<T> mat<T>::Zero(void);
static mat<T> mat<T>::Identity(void);
static mat<T> mat<T>::Translation(const vec<T>& t);
static mat<T> mat<T>::Rotation(const T& deg, const vec<T>& axis);
static mat<T> mat<T>::Perspective(const T& fov, const T& aspect, const T& nplane, const T& fplane);
static mat<T> mat<T>::Lookat(const vec<T>& eye, const vec<T>& center, const vec<T>& up);
static mat<T> mat<T>::Ortho(const T& l, const T& r, const T& b, const T& t, const T& n, const T& f);
static mat<T> mat<T>::Frustum(const T&l, const T& r, const T& b, const T& t, const T& n, const T&f);
static mat<T> mat<T>::Scale(const vec<T>& s);
static mat<T> mat<T>::Crossproduct(const vec<T>& v);
```

  - **Zero:** matrix consisting only of zeros
  - **Identity:** 4x4 identity matrix
  - **Crossproduct:** Given a 3D vector v, build a matrix M such that for a 3D vector u Mu = v x u, where x denotes the cross product. This matrix is useful for the implementation of the Rotation matrix.
  - **Translation, Rotation, Scale, Frustum** and **Ortho** are just like the OpenGL matrices described in

https://www.opengl.org/documentation/specs/version2.0/glspec20.pdf

  - **Perspective, Lookat** are described in

https://www.opengl.org/wiki/

- **Conversion from Quaternion to Matrix:** extend your matrix class with the following new methods and implement them:

```
const mat<T>& mat<T>::operator=(const quat<T>& q);
mat<T>::mat(const quat<T>& q);
```

These methods should generate a 4x4 rotation matrix from the quaternion q, see
http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/
Mind the notation! In particular, our implementation stores the real part in vector component 0! Include quat.h in mat.h.

- **Determinants & Matrix Inverses:** Extend your matrix class to include determinant and inverse computation. Specifically, add

```
T mat<T>::subdeterminant(size_t i, size_t j) const;
T mat<T>::determinant(void) const;
mat<T> mat<T>::adjoint(void) const;
mat<T> mat<T>::inverse(T* det = NULL) const;
mat<T> mat<T>::inverse_transpose(T* det = NULL) const;
```

  In **subdeterminant**, compute the determinant of a 3x3 matrix that is obtained by leaving row i and column j out of the 4x4 matrix.

  In **determinant**, develop the full 4x4 determinant from 4 subdeterminants. Mind the alternating signs. You can choose any development scheme (row, column, mixed) you like. If you are unfamiliar with this concept, the English Wikipedia entry on "Determinant" has an example for a 3x3 matrix.

  The **adjoint** (or adjugate) of a matrix is described in sufficient detail in the English Wikipedia entry on "adjugate matrix". Loosely speaking, adjoint(A) = determinant(a)*inverse(A).

  Using the **determinant** and the **adjoint** methods, implement the inverse method. If the argument **det!=NULL**, store the determinant in ***det**. In any case, if **determinant(A)==0,** return a zero matrix. The idea here is that if the user knows that the matrix is invertible, she/he would not bother with the argument det. If it is not clear whether the matrix is invertible, the argument det can be checked afterwards.

  Finally, implement **inverse_transpose**, which returns the transpose of the inverse matrix.

- **Test every method** you write in the main.cpp file. Submit your project including the test scenarios. Failure to submit adequate testing will result in loss of points for this assignment.

1.2 Part II

Part II will deal with the integration of part I in a simple OpenGL framework. In particular, the math implemented here will be used to navigate in a 3D scene and to manipulate a 3D camera. Part II will be posted shortly.