Documentation of Computer Graphics

Steven Tappert & Marvin Uwe Marken & Laura Mahlberg

May 15, 2018

HSB
Hochschule Bremen
City University of Applied Sciences

Contents

1 Abstract

This documentation shows and explains how the authors completed the given tasks in the course ”Computer Graphics”. The tasks are done in the IDE ”Qt Creator” with C++ as programming language. Main subject of the first tasks is to render 3D objects and subdivide the object faces into smaller faces later by using the Catmull-Clark Subdivision. After that the next tasks are about calculating Bézier Points to create and render bicubic Bézier surfaces and making a rotational surface out of a Bézier curve.

2 Quad-Mesh Connectivity

The first tasks are about creating fundamental classes that will be needed for further tasks, reading and rendering object files and providing a connected mesh at the end.

2.1 Rendering Quad Meshes

First needed are a vertex and a quad class.
Vertices are points in a two or three dimensional space that can be connected to each other to build the edges of an object (see figure 1). So for a vertex class the coordinates in a 3D environment are needed and for later purposes a vertex needs to know how many edges are connected to it, the so called ”valence”.
Quads (quadratic faces) are made of 4 vertices that build the face. Besides the 4 vertices (saved as IDs in the quad class) the quad also needs to know its adjacent quads, its neighbors. To determine what the outside of a face is we need to calculate the normal vector of the face. The normal vector is calculated by the cross product of the quads diagonals (see figure 2).
For rendering an object it is necessary to read its vertices and faces from a file. For these exercises it can be assumed that all faces are quads. The number of vertices and quads are variable depending on the object that is tried to load in. So instead of using static arrays it is required to use something dynamic like a QVector[1].
Now the object can be rendered. This will be done in two ways, as quadrilateral polygons to see the faces and how they reflect light (see figure 3) and as wireframe to see the vertices and edges.

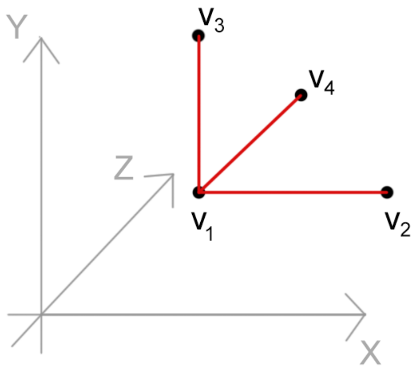


Figure 1: Vertex with some edges to other vertices

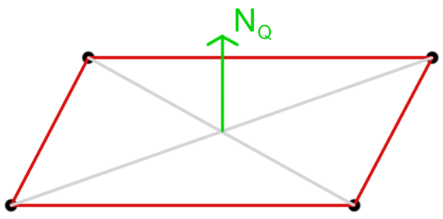


Figure 2: Quad with a normal vector in the center of it

2.2 Mesh Connectivity

In Problem 1 the created classes got variables like valence for the vertices or neighbors of the quads.

[1]http://doc.qt.io/qt-5/qvector.html

These need to be calculated now.
The valence of each vertex can be calculated by looking at every quad and see which vertex is next to the other (see figure 5) and see with how many other vertices that current vertex is connected. To prevent adding an edge twice to a vertex this needs to be checked as well.
For the neighbors of a quad it needs to be checked if a quad shares the same 2 vertices or edge with another quad, if it does it will be added to the current quad as its neighbor.

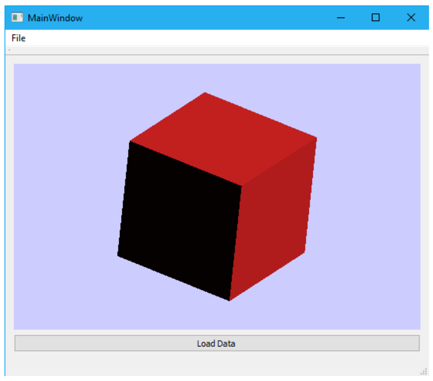


Figure 3: Render of an example cube

3 Catmull-Clark Subdivision

The goal of this exercise is to smooth objects, making them rounder by looking at nearby vertices of the altering vertex.

3.1 Catmull-Clark Subdivision Masks

For creating the subdivision mask there are some preparations needed before applying the subdivision in the end.

3.1.1 Quad midpoints

The first and easiest part is the calculation of the midpoints of the quads. To get the midpoint of a quad the average of all vertices of that quad is needed (see equation 1).

$$f = \overline{v_f} \tag{1}$$

So for quads every vertex needs to be divided by 4.

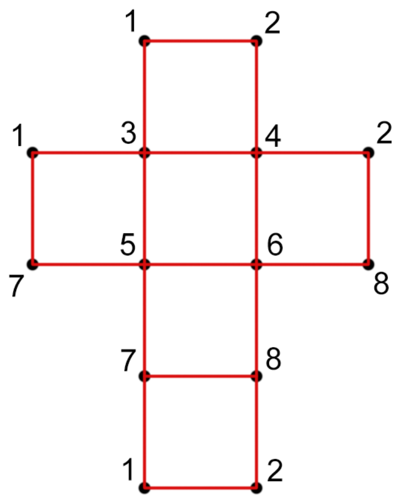


Figure 4: An opened cube with all vertex indices

3.1.2 Edge midpoints

The next thing needed for the subdivision mask are midpoints of every edge of an object. So for this kind of midpoint the average of both vertices of that edge and the average of the two quad midpoints next to the edge are needed.

$$e = \frac{1}{2}(\overline{v_e} + \overline{f_e}) \tag{2}$$

Because two averages of vertices were summed up, this sum needs to divide by two (see equation 2).

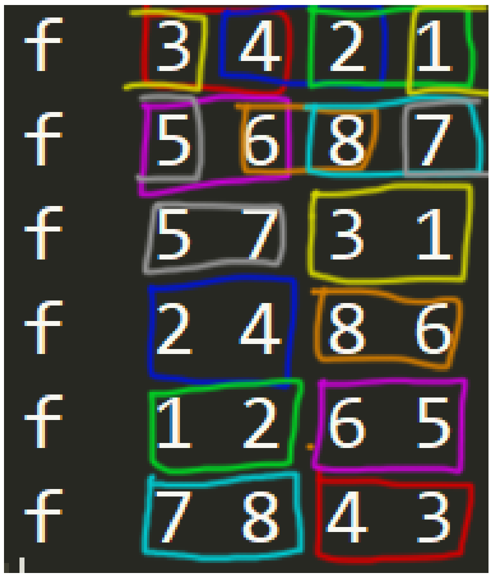
When going through all edges it needs to be prevented that an edge midpoint gets calculated twice due to indices may be flipped in another quad. For example Q_1 has vertices 1 2 3 4 and Q_2 has vertices 2156, the 1 and 2 are in both quads but in the second one it is the other way around and may not be detected as the same edge.

3.1.3 Alternative vertex mask

Now to get the alternative mask, the alternative vertices are needed. These are calculated as seen in equation 3.

$$v_{new} = \frac{1}{n_v}(4\overline{e_v} - \overline{f_v} + (n_v - 3)v) \tag{3}$$

n_v is the valence of the current vertex, $\overline{e_v}$ and $\overline{f_v}$ are the averages of all edge and face midpoints next to the vertex and v the original vertex.
After the alternative vertices are calculated they will replace the old originals.



```
f 3 4 2 1
f 5 6 8 7
f 5 7 3 1
f 2 4 8 6
f 1 2 6 5
f 7 8 4 3
```

Figure 5: A list of faces from the example cube.obj with edge building vertices shown (first two rows)

3.2 Recursive Subdivision

At the end all new vertices (face and edge midpoints and alternative vertices) get connected to new sub-faces, overwriting the old faces to restore the mesh connectivity. When the subdivision gets applied multiple times, the object gets more and more smoothed.

4 Bézier Surface

4.1 Bicubic Bézier Surfaces

Aside from the complex calculations there is not much more to it to construct a well-functioned Bezier surface. The surface is fundamentally a sum of Bezier curves. The calculation for this is to be found as part of assignment number five. To imagine a data structure which is able to organize every coordinate in a consistent and accessible way is challenging, and requires multidimensional arrays.

$$f(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij} B_i^m(s) B_j^n(t) \tag{4}$$

4.2 Reading Surfaces from File

To read and sketch a Bezier curve it was necessary to use a file format which is designed to do so. It was provided to create a JSON document where the coordinates can be read out for the development environment QT since QT shows a own library to create and process JSON documents.

```
if(firstChar == "{") {
        in.seek(0);
     Read frome JSON Dokument
     Put JSON Values in Array

                for(QJsonValue sliceVal : slice) {
                   Read points out of Dokument
        translate it into Array

                      static_cast<float>(pointObj["x"].toDouble()),
                      static_cast<float>(pointObj["y"].toDouble()),
                      static_cast<float>(pointObj["z"].toDouble())
                  };



           }else {
        could not read file;

}
```

5 Rotational Sweep Surfaces

5.1 Rotational Sweep Surfaces

The coefficients mentioned in the pseudo codes are in fact Bernstein polynomials. Knowing them allows to calculate any value on the curve from the start point till the end. For that the variable is given t [0,1], 0 being the start point and 1 the end. By dividing the current step by the total amount of steps one can compute the actual t.

5.2 Rotational Sweep Surfaces b.

6 Scene Design & Testing