LAB REPORT: LAB 3

TNM079, MODELING AND ANIMATION

Johan Beck-Norén
johbe559@student.liu.se

Tuesday 21st May, 2013

Abstract

This report covers the third lab in a series of six labs for the course TNM079 Modeling and Animation held at Linköping University. This lab covers the subject of splines and subdivision. A description of basic concepts in spline and subdivision theory is given in the beginning of this report, followed by assignments completed using this theory in practice to perform subdivision of curves and meshes. Techniques implemented include curve- and mesh subdivision, local evaluation of the analytical spline, and an adaptive subdivision scheme based on local vertex curvature. The results show that subdivision is a useful tool in computer graphics and can be coupled with different mesh structures with good results.

1 Introduction

Subdivision schemes for curves and surfaces are well established techniques in computer graphics. Because of their nature, they are fast to compute, numerically stable and operations only affect the curve locally. In this section some basic spline- and curve theory will be covered.

1.1 Curves and splines

Describing curves in computer graphics is an inherently difficult problem. Since computers use discrete schemes as opposed to infinite descriptions, a curve is described as a parametric curve defined by a set of coefficients and basis functions. The parametric curve *p(t)* can be described as

$$p(t) = \sum_{i=0}^{n} c_i t^i \tag{1}$$

where c_i is the i:th coefficent and t^i is the i:th polynomial basis function. These polynomials are very easy to compute and are differentiable everywhere. Complex shapes can be achieved with low order curves by combining these lower order curves.

1.2 Piecewise curves and continuity

As discussed in the previous section, multiple low-order curves can be combined to form what looks like a single line with complex shape. Because the order of the basis functions are low the curves will still be fast to compute. The disadvantage however is that the different curve parts must be connected explicitly and boundary conditions must be handled with care in order to make the transition between curve segments sufficiently smooth. More specifically

the smoothness is defined as the continuity, C^n, where n is the number of derivatives at a joint between curves. C^{-1} means that the endpoints of adjacent curves are maybe not even connected, C^1 means that the curves share the same first derivative and so on.

1.3 Bézier curves

The use of Bézier curves is common practice in computer graphics, vector graphics and many other applications. The curve is created by interpolating between control points. A simple example of a linear Bézier curve, given control points p_0 and p_1, is given by

$$p_1 = (1-t)c_0 + tc_1, t \in [0,1]. \tag{2}$$

Although the result from this example will look like a single straight line, the curve points are still interpolated and is therefore equivalent with linear interpolation between the control points. This is not generally the case though. A more common implementation is called quadratic Bézier curves, which uses three control points for the *approximation* of curve points.

1.4 Convex combinations

A *convex combination* is a linear combination of data points where all coefficients sum up to one and are non-negative. A set containing all convex combinations constitutes the *convex hull* of the given data points. We say that a convex combination that meet the criteria above is numerically stable, since small perturbations in the input result in small perturbations in the output.

1.5 B-splines

As mentioned in section 1.1, we use something called basis functions. For the linear case discussed earlier we had one basis function for each coefficient on the interval [0,1]. Now, consider a cubic basis function on the interval [-2,2].

$$B_{i=0,3}(t) = \frac{1}{6}\begin{cases} (t+2)^3, -2 \leq t < -1 \\ -3(t+1)^3 + 3(t+1)^2 + 3(t+1) + 1, -1 \leq t < 0 \\ 3t^3 - 6t^2 + 4, 0 \leq t < 1 \\ -(t-1)^3 + 3(t-1)^2 - 3(t-1) + 1, 1 \leq t \leq 2 \\ 0, otherwise \end{cases} \tag{3}$$

This basis meets the criteria stated in section 1.4 and is therefore stable. For a set of coefficients we can associate a basis function with each coefficient according to equation (1), where t^i represents a basis function. The basis function in equation (3) is called the cardinal cubic B-spline basis function. It is evaluated in the same fashion as a Bézier curve, as a sum of the products between coefficients and their corresponding basis functions.
An interesting property of the B-spline is construction by convolution. A B-spline of degree d can be found by recursive convolution, by convolving B_{d-1} with B_0 (4).

$$B_d(t) = \int B_{d-1}(s)B_0(t-s)ds \tag{4}$$

Another interesting property, which connects to the section on subdivision, is that B-splines are refineable.

$$B_d(t) = \frac{1}{2^d} \sum_{i=0}^{d+1} \binom{d+1}{i} B_d(2t - i) \tag{5}$$

In this refinement equation, equation (5), it is stated that a B-spline of degree d can be described as a linear combination of dilated and translated copies of itself. For every refinement, the spline's continuity is increased by one degree. For a cubic B-spline, the refinement is

$$B_3(t) = \frac{1}{8}(1B_3(2t) + 4B_3(2t-1) + 6B_3(2t-2) + 4B_3(2t-3) + 1B_3(2t-4)). \tag{6}$$

1.6 Subdivision of spline curves

As seen in the previous section, we can apply refinement to a spline and receive new basis functions. When the basis' change the coefficients must also change, and finding these new coefficients is what subdivision is all about. We will use vector notation for the basis functions and the coefficients. Let the basis functions $[B_n(t), B_n(t-1), B_n(t-2), ..., B_n(t-n)]$ be denoted $\vec{B}$ and the coefficients $[c_0, c_1, c_2, ..., c_n]$ be denoted $\vec{C}$. The spline curve then becomes

$$p(t) = \vec{B}(t)\vec{C} \tag{7}$$

After dilating the basis functions, as described in the previous section, we can use the new basis as support. In equation (8), $\hat{S}$ denotes the coefficients in matrix form populated with values provided by equation (5).

$$\vec{B}(t) = \vec{B}(2t)\hat{S} \rightarrow p(t) = \vec{B}(t)\vec{C} = \vec{B}(2t)\hat{S}\vec{C} \tag{8}$$

The result of this refinement is the same curve as before, only described by B-splines whose control points are spaced twice as dense. When evaluation by successive subdivision, as opposed to analytical evaluation of a spline, there are only two rules for the new coefficients. In equation (9) c_i' denotes the re-weighted value for the i:th coefficient and $c_{i+\frac{1}{2}}$ denotes the new coefficient corresponding to the refined basis function created between the coefficients c_i and c_{i+1}.

$$\begin{aligned} c_i' &= \frac{1}{8}(1c_{i-1} + 6c_i + 1c_{i+1}) \\ c_{i+\frac{1}{2}}' &= \frac{1}{8}(4c_i + 4c_{i+1}) \end{aligned} \tag{9}$$

1.7 Boundary constraints

There exists an exception for the coefficient rules in equation (9), namely boundary constraints. If we are evaluation the start point with coefficient c_0, we cannot evaluate c_{i-1}. Similary for the end point we cannot evaluate c_{i+1}. This is solved by simply ignoring the boundary coefficients in the subdivision scheme by keeping their old values.

$$\begin{aligned} c_0' &= c_0 \\ c_{end}' &= c_{end} \end{aligned} \tag{10}$$

1.8 Mesh subdivision

Mesh subdivision is the process of subdividing a coarser, often low-polygon, 3D mesh to achieve a mesh with a more smooth surface. More specifically it is an iterative process to divide polygonal faces into smaller faces and calculating new vertex positions. The smooth surface can be calculated with the original mesh as the limit of this iterative process. Each subdivision step will produce more faces that in turn will better approximate the smooth surface. Unfortunately, the theory described above only carries over to 2D. When applying subdivision to meshes, other schemes have to be used.

In this section and in the following sections we introduce a new term, *valence*. Valence is the number of incident edges to a vertex. An *ordinary vertex* has a special valence which varies depending on the mesh type, 4 for quad meshes and 6 for triangle meshes. If a vertex's valence differs from this number, it is called an *extraordinary vertex*.

1.8.1 Loop subdivision

Loop subdivision is an approximation scheme for triangle mesh subdivision and is a generalization of spline curve refinement to irregular meshes. It is C^2 continuous at regular vertices and at least G^1 everywhere else. Since we are using a half-edge mesh structure in this lab we have quick and easy access to vertex- and face neighbourhoods, which makes the loop subdivision scheme easy to implement. Assuming a triangle mesh, each triangle is split into 4 new triangles as seen in figure 1. New vertex positions are calculated from the weighted averages of the old vertices according to figure 2. In figure 2, both internal and boundary vertex weights are shown. We calculate β in equation (11) using the valence k.

$$\beta = \begin{cases} \frac{3}{8k}, k > 3 \\ \\ \frac{3}{16}, k = 3 \end{cases} \tag{11}$$

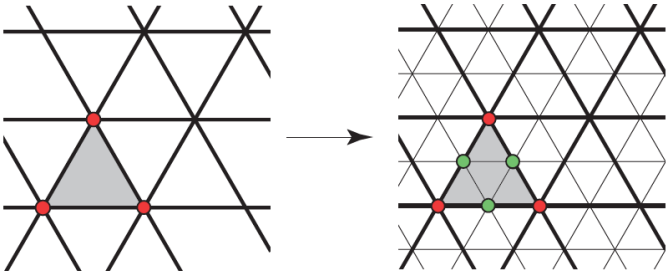

Figure 1: Triangle refinement using Loop subdivision scheme.

2 Assignments

2.1 Implementing curve subdivision

For this assignment we use the rules described in section 1.6, namely equations (9) and (10). We iterate through the container holding all the old coefficients and calculate new weighted positions as well as inserting new coefficients between every other old coefficient. Boundary

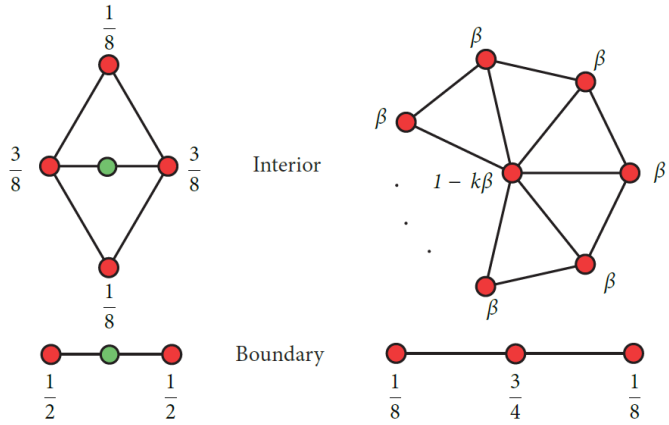

Figure 2: Weights for new vertex positions using Loop subdivision with valence k.

constraints are handled as discussed in section 1.7. The start point and end point are handled before and after the loop respectively.

Algorithm 1 Curve subdivision

1: $newCoeff.append \leftarrow oldCoeff.at(start)$
2: $newCoeff.append \leftarrow (4 * oldCoeff.at(start) + 4 * oldCoeff.at(start + 1))/8$
3: **for** All internal old coefficients **do**
4: $\quad c = (1 * oldCoeff.at(previous) + 6 * oldCoeff.at(current) + 1 * oldCoeff.at(next))/8$
5: $\quad cHalf = (4 * oldCoeff.at(current) + 4 * oldCoeff.at(next))/8$
6: $\quad newCoeff.append \leftarrow c$
7: $\quad newCoeff.append \leftarrow cHalf$
8: **end for**
9: $newCoeff.append \leftarrow oldCoeff.at(end)$

The algorithm above will, from a curve using N points, produce a curve with $2N - 1$ points. Algorithm 1 was implemented in the method Subdivide in the class UniformCubicSplineSubdivisionCurve.

2.2 Implementing mesh subdivision

The code skeleton given in this lab already had an algorithm for adding new vertices along edges as seen in figure 1. The task was to implement Loop subdivision, so the remaining problem was to implement the algorithms for calculating the new weighted vertex positions as seen in the left portion of figure 2.

$$v_{new} = \frac{1}{8}(3v_0 + 3v_1 + v_2 + v3) \tag{12}$$

We also need to compute new positions for the old, already existing, vertices. This is done according to the right portion of figure 2, using equation (11). For each vertex, we find its 1-ring neightbourhood N and accumulate the weighted position values.

$$v_{i,new} = (1 - k\beta)v_i + \sum_{j \in N(i)} \beta v_j \tag{13}$$

The algorithm for calculating new positions for vertices created along edges was implemented in the method EdgeRule, and the algorithm for calculating new weighted positions for already existing vertices was implemented in the method VertexRule. Both methods can be found in the class LoopSubdivisionMesh.

2.3 Localized evaluation of the analytical spline

The goal of this assignment was to localize the evaluation of the cubic spline. The naive implementation present in the lab skeleton evaluates all control points, regardless if they have a supporting basis function or not. By finding the control points that are supported by basis functions for a given point t, we can localize the evaluation of the curve. For the cubic spline, this amounts to at most 4 basis functions for each control point.

Algorithm 2 Localized evaluation of analytical spline

1: ControlPoint $p = floor(t)$
2: $valueSum = coeffs.at(p) * GetBSplineValue(p, t)$
3: **if** $p > 0$ **then**
4: $\quad bVal = GetBSplineValue(p - 1, t)$
5: $\quad valueSum+ = coeff.at(p - 1) * bVal$
6: **end if**
7: **if** $i < end - 1$ **then**
8: $\quad bval = GetBSplineValue(p + 1, t)$
9: $\quad valueSum+ = coeff.at(p + 1) * bVal$
10: **end if**
11: **if** $i < end - 2$ **then**
12: $\quad bVal = GetBSplineValue(i + 2, t)$
13: $\quad valueSum+ = coeff.at(i + 2) * bVal$
14: **end if**
15: **return** $valueSum$

The pseudocode in algorithm 2 can be found in the method GetValue in the class UniformCubicSpline.

2.4 Adaptive mesh subdivision by local mean curvature

The intent of subdivision is to achieve a more smooth mesh. This will naturally result in a higher polygon count. While this will generate a nice smooth looking mesh, it becomes very expensive very fast in terms of face count. It is therefore worth looking at methods which to a degree compromises smoothness in favour of face count. The area of adaptive subdivision consists of classifying which mesh regions to subdivide, and which to ignore. The classification decision can depend on just about anything. In this report we chose to examine an adaptive mesh subdivision scheme which takes local mean curvature into consideration. We created a scale that is normalized between zero and the max absolute mean curvature value of the mesh,

and use a threshold to decide whether a region will be subdivided or not. This resulted in areas with high curvature would be subdivided, while locally flat areas would not be.

The code for this assignment can be found in the method Subdividable in the class StrangeSubdivisionMesh.

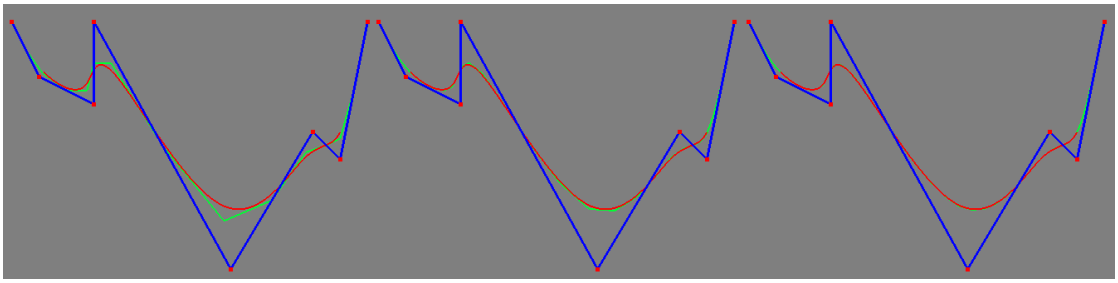

Figure 3: Curve subdivision. Analytical spline in red. Green curve subdivided once, twice and three times from left to right.

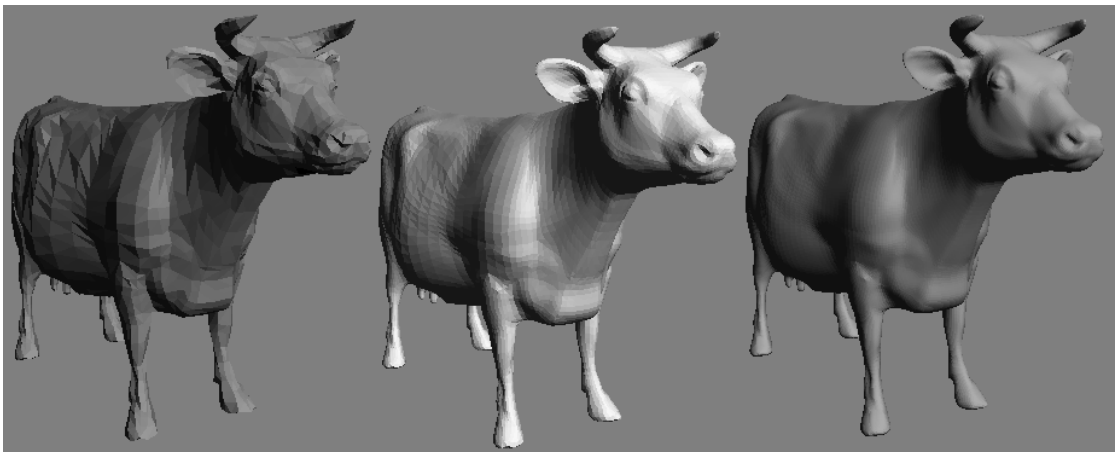

Figure 4: Mesh subdivision. Original mesh to the left, one subdivision middle, and two subdivisions right.

3 Results

3.1 Curve subdivision

The curve subdivision gives a satisfactory result, but a few subdivisions are needed before the curve approximates the analytical spline sufficiently. The results can be seen in figure 3 with the analytical spline in red and the approximated curve in green.

3.2 Mesh subdivision

The implementation for mesh subdivision works well and produces more smooth meshes for every subdivision. When implementing equation (12), the code had to be written

$$v_{new} = 3 * (v_0 + v_1) * 0.125 + (v_2 + v3) * 0.125 \tag{14}$$

instead of using the fractions as seen in equation (12). This reason for this is numerical errors present in computers.

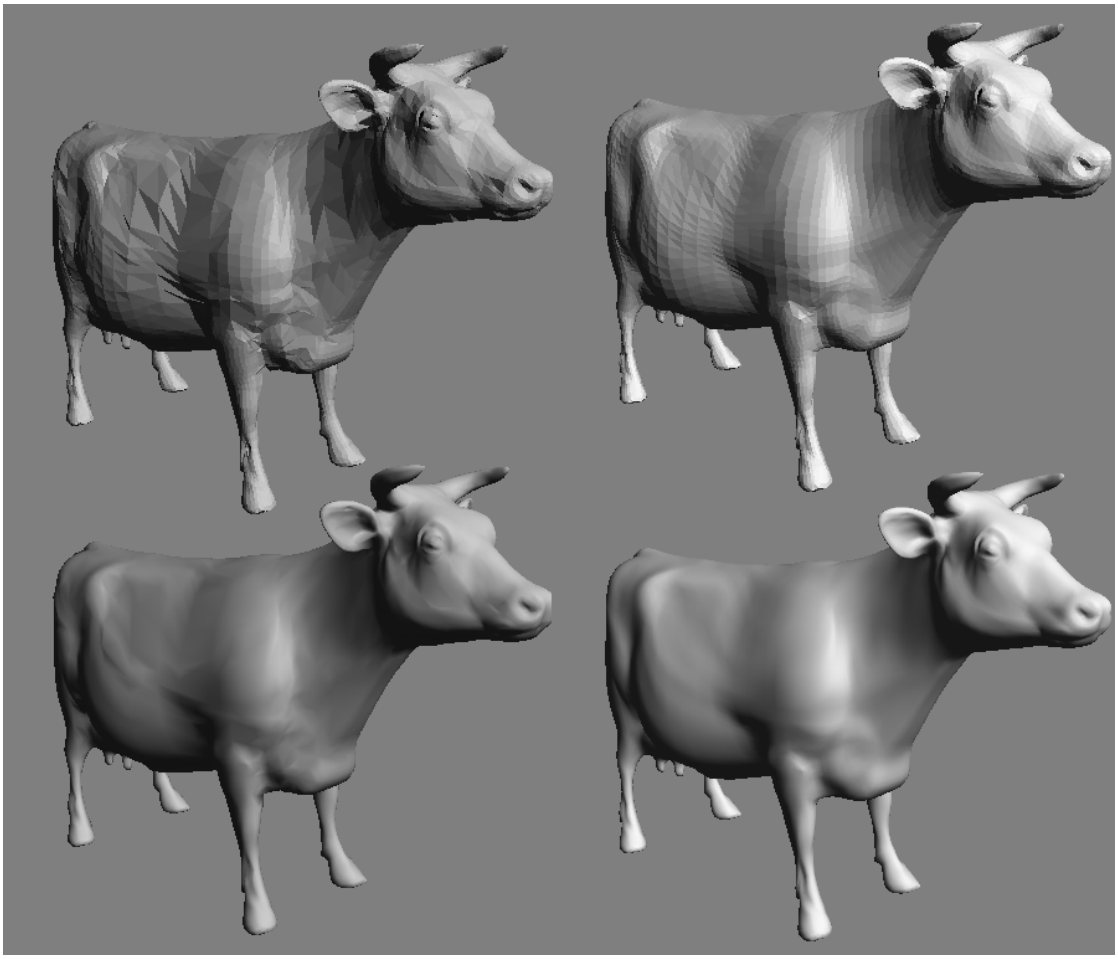

Figure 5: Adaptive mesh subdivision by local mean curvature. Adaptive subdivision top and bottom left without and with a vertex lighting model. Regular subdivision top and bottom right without and with a vertex lighting model.

3.3 Localized evaluation of the analytical spline

The localized evaluation seems to be working fine and produces the exact same visual result as the previous, naive, implementation.

3.4 Adaptive mesh subdivision by local mean curvature

The scheme implemented for adaptive mesh subdivision by local mean curvature works reasonably well. In figure 5 a comparison is shown between our adaptive subdivision and a regular straight-forward subdivision which does not take curvature into consideration. We see that the detailed areas around the cow's horns, face and hooves are subdivided, while the flatter areas along the stomach, neck and face are left untouched. The figure also demonstrates how our scheme differs by showing the two models with a vertex lighting model applied. The results are surprisingly good considering the difference in triangle count between the two models.

4 Conclusion

From the theory presented and the assignments completed in this report, we see that curve subdivision is rather easy to implement. Only a few subdivision iterations are needed before the subdivided curve approximates the analytical curve rather well.

For mesh subdivision using Loops subdivision scheme we benefited from using a half-edge mesh structure when performing mesh subdivision due to quick and easy access to the local vertex neighbourhoods.

The localization of the analytical spline produces the same visual results as the previous naive implementation. But, we have cut down on the number of coefficient accesses needed since we now only access the coefficients with a supporting basis present.

The implemented adaptive mesh subdivision scheme works well. We found that a threshold of 1% on the scale from zero mean curvature to the max absolute mean curvature value of the mesh worked well. This may seem low, and it is. An improvement to this scheme would be to take the curvature distribution of the mesh into consideration as well. Otherwise very large or very small curvature values may skew the results in a bad way.

5 Lab partner and grade

The assignments in this lab were completed together with Hans-Christian Helltegen (hanhe945). I have completed all assignments marked with (*), (**) and (***), therefore I should qualify for grade 5.