CS 633 3D Computer Animation

Solution Set - HW 3 (40 points)

1. "Cubic interpolation" is a popular path smoothing technique (slides 38-39 of notes: Interpolating Values III). The approach is as follows: for each point P_i, construct a cubic curve $P(t)$ to interpolate P_{i-2}, P_{i-1}, P_{i+1} and P_{i+2} at $P(0)$, $P(1/4)$, $P(3/4)$ and $P(1)$, and then use the value of $P(1/2)$ to adjust P_i. The new location of P_i is defined as

$$P_i' = (P(1/2) + P_i)/2.$$

Why wouldn't we use $P(1/2)$ as the new location of P_i directly? (10 points)

Sol.

Since P_{i-2}, P_{i-1}, P_{i+1} and P_{i+2} all carry some error, $P(1/2)$ computed from these points carries error as well. Therefore, it makes no sense to replace P_i with $P(1/2)$, especially when the error carried by $P(1/2)$ might even be bigger than that of P_i. Using the average of P_i and $P(1/2)$ as the new location of P_i, in the best case, can cancel out the errors carried by these terms and, in the worst case, would still reduce by one half of the difference of their errors.

2. Using the "*Shortest path*" approach to find a path from a start point (S) to a destination point (D) on a polygonal surface mesh, one needs to check that, after unfolding of all the faces, if the line segment that connects S and D lies completely inside the unfolded faces. How should this (the unfolding process) be done efficiently? (10 points)

Sol.

Exhaustively testing all possible unfolding combinations is certainly possible. But this is a very costly approach. A better approach is to run a *greedy algorithm* to find an approximation of the optimal path (called the *greedy path*) first, and then unfold the polygonal surface mesh as follows. The idea is that if a shortest path exists, then edges of the shortest path must be inside the faces adjacent to the greedy path.

1. Sort the edges of the *greedy path* and name them E_i, $i = 1, 2, \ldots, n$. Each E_i is considered a directed edge (P_i, P_{i+1}) with $P_1 = S$ and $P_{n+1} = D$.
2. For each E_i in the list, cut the polygonal mesh along edges adjacent to E_i (include E_i itself) in the following fashion:
   - 2.1 For E_1 (E_n), cut each edge adjacent to the start point S (end point D) from S (D) to the other endpoint; // see Fig. 1
   - 2.2 For each of the remaining E_i, $i = 2, 3, \ldots, n - 1$, cut E_i from P_i to P_{i+1}; // see Fig. 2
   - 2.3 For an edge not adjacent to any E_i (excluding edges added during the triangulation process), cut the edge from one endpoint to the other endpoint.

3. Unfold the faces of the polygonal mesh as follows:

3.1 For E_1 (E_n), unfold each face that contains E_1 (E_n) about the edge that is not adjacent to the start point *S* (the destination point *D*) so that it is co-planar with a face that contains E_2 or adjacent to E_2. In the second case, repeat this process until a face that contains E_2 is reached; // see Fig. 3

3.2 For each of the remaining E_i, *i* = 2, 3, . . . , *n* - 1, unfold the face that contains E_i about an edge adjacent to P_{i+1} so that it is co-planar with a face that contains E_{i+1} or adjacent to E_{i+1}. In the second case, repeat the same process as in 3.1. Note that all the faces considered here do not have dangling adjacent faces. // see Fig. 4

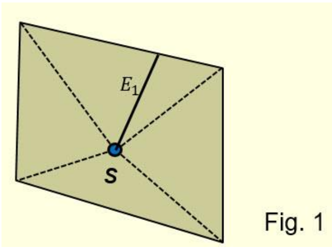


Fig. 1

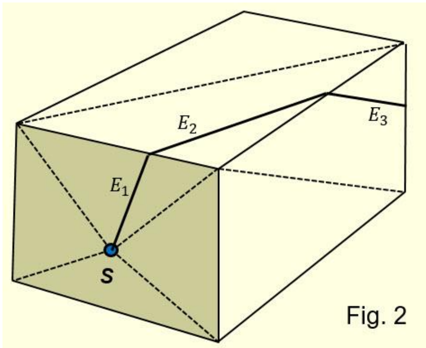


Fig. 2

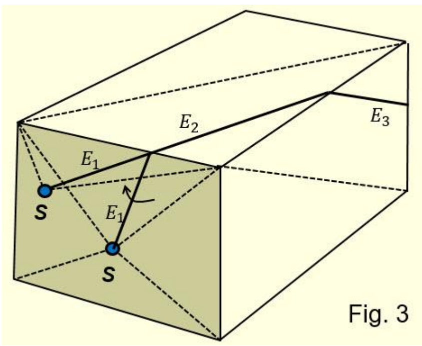


Fig. 3

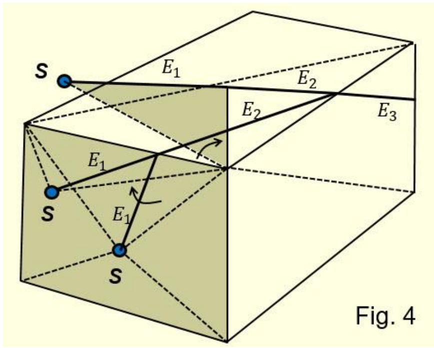


Fig. 4

3. **A** = (*x*, *y*, *z*) and **B** = (*a*, *b*, *c*) are two points of a bicubic Bezier surface patch **S**(*u*, *v*), 0 ≤ *u*, *v* ≤1. A path along the surface **S**(*u*, *v*) from **A** to **B** can be constructed as follows: find the points (*s*, *t*) and (*p*, *q*) in the parameter space of **S**(*u*, *v*) such that **A = S**(*s*, *t*) and **B = S**(*p*, *q*), then map the line segment that connects (*s*, *t*) and (*p*, *q*) onto the surface. The resulting curve is a good path from **A** to **B**. How would you find (*s*, *t*) and (*p*, *q*)? (10 points)

Sol.

Using *midpoint subdivision method*. The idea is as follows. We will illustrate the process for point **A** only. The process for point **B** is similar.

Let **P** = {$\mathbf{P}_{i,j}$ | 0 ≤ *i*, *j* ≤ 3 } be the control point set of **S** and let

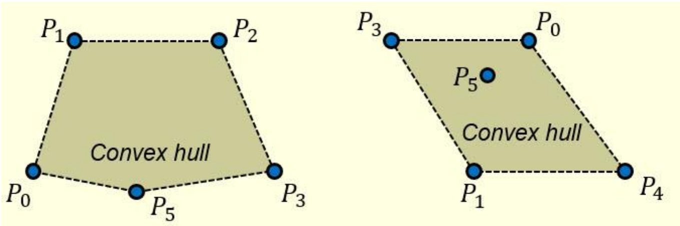
$$a = 0; \quad b = 1; \quad c = 0; \quad d = 1.$$

(a and b here have nothing to do with the coordinates of **B**. Then use the following algorithm to find the parameters (s, t) of **A**.

```
while ((b - a > ϵ) and (d - c > ϵ)) do {
    perform midpoint subdivision on P to get
        E = { E_{i,j} | 0 ≤ i, j ≤ 3 }    /* c-pts for [a, (a + b)/2] × [c, (c + d)/2]*/
        F = { F_{i,j} | 0 ≤ i, j ≤ 3 }    /* c-pts for [(a + b)/2, b] × [c, (c + d)/2]*/
        G = { G_{i,j} | 0 ≤ i, j ≤ 3 }    /* c-pts for [a, (a + b)/2] × [c + d)/2, d]*/
        H = { H_{i,j} | 0 ≤ i, j ≤ 3 }    /* c-pts for [(a + b)/2, b] × [(c + d)/2, d]*/
    if A is in the convex hull of E { // how do we know?
        b ← (a + b)/2;   d ← (c + d)/2;   P ← E; }
    else if A is in the convex hull of F {
        a ← (a + b)/2;   d ← (c + d)/2;   P ← F; }
    else if A is in the convex hull of G {
        b ← (a + b)/2;   c ← (c + d)/2;   P ← G; }
    else {
        a ← (a + b)/2;   c ← (c + d)/2;   P ← H; }
    }
s ← (a + b)/2;   t ← (c + d)/2;
```

Note:

(1) The *convex hull* of a set of given points is the smallest convex set that contains the given points. If the number of given points is finite, then the convex hull is basically the set of all linear combinations of the given points. See the following figure for a 2D example of the convex hulls of five control points.



(2) For the definition of *midpoint curve subdivision*, please see slide 59 of "Cubic Bezier Curves and De Casteljau algorithm (recurrence formula)" on the left side of course website "Related Course Materials". *Midpoint subdivision* of bi-cubic surface patch is a simple extension of the curve case.

4. In 3D free-form deformation, after the manipulation of the 3D coordinate grid, the deformed position of a vertex of the object is determined through a *trivariate Bezier interpolation process*. What is the reason in doing so (in your opinion) ?   (10 points)

Sol.

Note that the shape of a Bezier curve reflects the shape of its control polygon and the shape of a Bezier surface patch reflects the shape of its control mesh. Similarly, the shape of a 3D Bezier solid reflects the shape of its control grid.

In our case, the initial shape of the trivariate Bezier solid coincides with the coordinate control grid (including all the interior points) because vertices of the coordinate grid are co-linear. Hence the trivariate Bezier solid contains the given object as a subset.

After the manipulation of the 3D coordinate grid, the trivariate Bezier solid has a new shape because the coordinate grid, acting as its control grid, changed. As pointed out above, the new shape of the trivariate Bezier solid reflects the new shape of the coordinate grid. Hence the new shape of the object whose vertices are points of the new trivariate Bezier solid would also reflect the new shape of the coordinate grid.