CMPS 161:

Final Project Report: Free-Form Deformation

Thuc H. Nguyen
Email: thnguyen@ucsc.edu
Winter 2011

OpenGL Animation with the application of Free Form Deformation

Thuc H. Nguyen, University of California at Santa Cruz

CMPS 161: Winter 2011

Abstract:

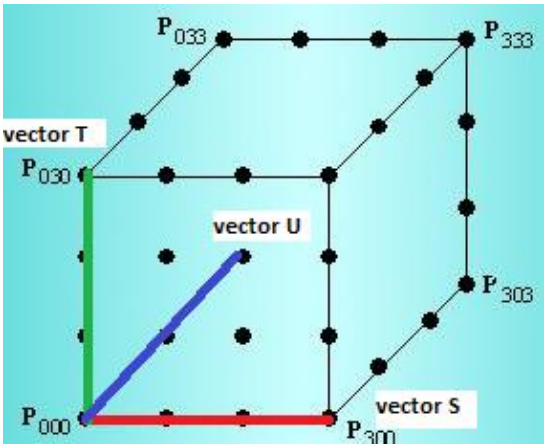
Computer animation is the technique of rapid displaying a sequence of images in either two or three dimensional artwork. Certainly there are many different approaches in creating animated models, some of which required the modeling of positions relative to the object as a form of illusion of movements. In computer graphics, the use of global transformation is applied toward simple objects such as cube, cylindrical cone or sphere to obtain the affect of moving shapes. First, the transformations are applied on the coordinates of the rendering shape; the object is then being rendered "off screen" using the new transformed set of coordinates. Lastly the newly rendered shape is moved onto the screen while another set of transformation is being applied to the off screen content. The global transformations, which include a vast number of methods such as tapering, twisting and bending along the primitive transformations such as translating, rotating and scaling, are sufficient for obtaining a highly animated motion picture. However for non-primitives shapes such as 3D models of animals, human and etc…the use of only global deformation is an insufficient approach as it will place a constraint on the ability to create many different animated movements on a single object as well as the computing time will exceed unreasonably. While transformations are applied to each vertex on the object, the mentioned constraint will increase as more vertices are present. Free Form Deformation was introduce to greatly reduced the computing time while granting more possible movements of the objects with of any vertex sizes. With the use of Free Form Deformation, the animation aspect in computer graphics was able to explore more possibilities while limiting the resources in term of computation time.

Introduction:

Free Form Deformation (FFD) is a quality approach toward animating two or three dimensional objects due to its ability to adapt to any models without having to recode any aspect of FFD. Secondly, the global transformation in FFD is now apply toward the whole object of any shape rather than to only any portion of the object in the case when FFD was not applied, which creates a more realistic looking motion picture. Free Form Deformation uses the technique of applying a cubic frame (preferably 4x4x4) to "wrap" around the object. Global transformation is only

applied on this cubic frame which in turn transformed the object. The visualization aspect of this technique can be compared to the string puppet theory. In a string puppet play, in order to portrait the character's movement, the puppet actor applies upward forces (movements) to the strings connected to different parts of the puppet. In animation using FFD, the same theory is applied however transformations (movements) can be applied to any part in any direction of the object through the use of the cubic frame, and not just a single vertical upward movement. In order to create the connections between this cubic box and any given object embedded in this box, a series of mathematic calculations were utilized.

The program starts off with the initialization of set of control points which formed the 4x4x4 cubic frame. This set of control points is a primary underlying concept of how Free Form Deformation works. Later on, all global transformations are applied directly on the points which affect the positions of all its neighboring points in order to create a realistic animation. Once these control points are established, a local coordinate system is imposed on the 3D model. This local coordinate system is determined entirely based on the location and spacing of the control points, mainly the points locating at the lower-most and left-most corner (P_zero vertex), the lower-most and right most corner, the upper-most and left-most corner, and lastly the lower-most, left-most and inner-most corner. These points form vectors with direction pointing out from the P_zero vertex called vector S, vector T and vector U, respectively based on the listing above. These primary vertices are not necessarily mutually perpendicular, however most online examples are using this form of frame.



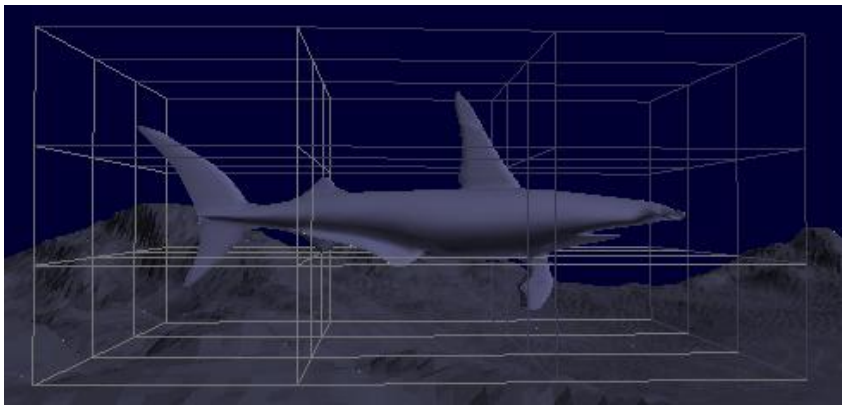
Once the local coordinate system for the cubic frame has been established, the program utilizes the vectors (S, T, U) to obtain the position of the object relative to this box. In another word, since the object is encapsulated by this cubic frame, all of its (the object) coordinates can be converted so that they are locally to this frame. Using a formula, the program will iterate through all of the object's vertices and convert them from x,y,z coordinates to s,t,u coordinates, respectively. These coordinates represent the location of the object in correspondent to the location of the frame such that at any point (vertex) on the object, s,t,u will be from zero to one. The s,t,u values from zero to one of a point indicate its relativity to this frame; that is the point is

enclosed inside the frame . We will go into more detail of the local coordinate in the later part of this report.

There are many algorithms to implement Free Form Deformation from here on, and this project is implemented with the "Sederberg and Parry's Free Form Deformation". From top view, this algorithm utilize the established s,t,u coordinates of the object to find the relative position of this object after the control points on the frame have been transformed. Once a transformation is applied onto the frame, the values of the local coordinates will be changed thus s,t,u coordinates of the object will also be changed. From here the Sederberg and Parry's Free Form Deformation algorithm will determine the new position of the object by plugging the s,t,u into the "Trivariate Bézier function". This function will produce a set of x,y,z coordinate for every set of s,t,u coordinate that was fed into it. Subtract the current x,y,z coordinates by this newly acquired coordinate will provide the program with the amount of changes that should be made to the object so that it is animated in corresponding to the frame of control points. Free Form Deformation can be used in two ways, once of which is called direct manipulation (single points). This form of FFD allows the user or the programmer to move any control points relative to the object, this in turn will deform the location of the object corresponding to that control points. The second method takes advantage of the possibilities which FFD provide by transforming the entire set of control points to form a global transformation on the object

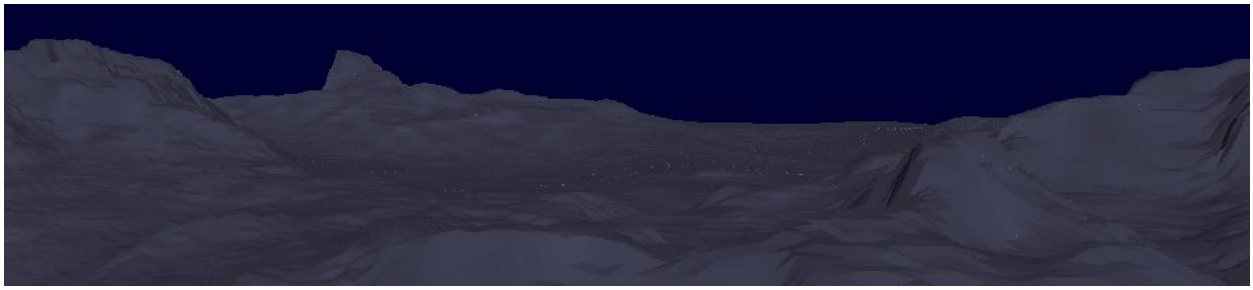
Related Work

This project implemented a sequence of animation using the mentioned Sederberg's and Parry's Free Form Deformation algorithm. I decided to continue working on our class's first programming assignment since I was always fascinated by the look of the shark. In the class's first program, we were assigned to render a shark by obtaining the given x,y,z coordinates and its polygonal connections from two different input file. Using the animation approach of FFD, the project transformed a lifeless 3D shark model into something that could swim and do many other motions depending on the choice of the programmer. Changing from one motion to another only will require minor changes on the programmer's end.



The image above was taken from the actual project which portraits a fully rendered shark enclosed in the frame box. At each fork of this cubic box (frame) is a control point. Some of which are get transformed by the FFD transformation function. Once the transformations are applied onto the frame, the new coordinates of those control points are fed to the Trivariate Bezier function for further processing which causes the shark, enclosed within this frame, to move corresponding to the transformation of the cubic frame. Once the Free Form Deformation was implemented, the next part is to correctly translating the control points in such a way that it will successfully mimic the body movement of the shark. Since the control frame which contain the control points forms a 4x4x4 cubic box with the total of 64 control points and the body of the shark is horizontal aligned, sub-dividing the body into 4 sections from head to tail helps maintaining the physically abide swimming motions. First approach at creating this swimming motion was using the global bending transformation, however, the body did not move as wanted thus I switched to the simpler approach of global translation. As the head portion of the shark swings side to side, upper and lower mid section also swing at lower amplitude to maintain the shark's original form. The tail portion is the last to swing yet it has the highest swing amplitude. In term of swinging speed, the top half of the shark including the head, swings at a slightly slower frequency compared to the bottom half. These choices of motion were determined by close observation of the documentaries of the actual Hammer Head shark.

The extra that was added to this project was the terrain mapping of the ocean's bottom. Using image mip-mapping, the program utilizes two different images, one was for surface mapping of the ocean bottom while the other was used for obtain the height field of the terrain to create a depth of field look. After the second image was read in, the height field was created by iterating through every bits of this image and recording the gray value of the image. This is a built-in function call in openGL [*qGray(map.pixel(widthOfImage, HeightOfImage))*] that will return a float value which depends on the gray-intensity of the image. This method is useful in creating mountains and valley since the gray value either increasing or decreasing at each image bit in a gradual rate which creates a more realistic looking terrain. For more effective normal calculation of the terrain's normal, the program rendered the mesh with triangle strips, this way normal calculation requires less computation time, and less debugging problems. To save computation time further, only three 512x512 meshes were created then scaled to a larger value, thus the terrain is distorted by some margin, especially the height field while still maintaining its realistic appearance to some degree.



Technical Detail

Let's go into more detailed of the method and technique required for creating the Free Form Deformation. Once the program has determined the current location of the shark, a function will be called to form the local space, namely the P zero and the vectors S,T,U mentioned in the introduction. Vector S was created by subtracting vertex at P_zero from the vertex at S-location (lower, right-most corner), and so forth for the T and U vectors. Online documentations use U, V, W to represent S, T, U respectively. Once the frame work above is obtained, the program calls another function to obtain the local "s,t,u" coordinates (not vectors) of the shark using the formula below. Again, online documentations refer to these (s,t,u) coordinates as (u,w,v) coordinates respectively.

$$u = \frac{(\vec{v} \times \vec{w}) \cdot (\mathbf{P} - \mathbf{O})}{(\vec{v} \times \vec{w}) \cdot \vec{u}}$$

$$v = \frac{(\vec{u} \times \vec{w}) \cdot (\mathbf{P} - \mathbf{O})}{(\vec{u} \times \vec{w}) \cdot \vec{v}}$$

$$w = \frac{(\vec{u} \times \vec{v}) \cdot (\mathbf{P} - \mathbf{O})}{(\vec{u} \times \vec{v}) \cdot \vec{w}}$$

The cross product of the U,V,W was self-explanatory, and $\mathbf{P}$ is representation of a vertex of the shark, thus this formula iterates through every single vertex of the shark and creates as many (s,t,u) coordinates as the shark's vertices. $\mathbf{O}$ is the representation for P_zero. The shark has the total of 2560 (x,y,z) coordinates thus the function above will form 2560 (s,t,u) coordinates corresponding to each shark's vertex. This calculation considered vital since it introduces a set of local coordinates to the shark along with the existing global (x,y,z) coordinates. Once this set of local coordinates is determined, each vertex ($\mathbf{P}$) on the shark is recalculated using the this formula…

$$\mathbf{P} = \mathbf{O} + u\vec{u} + v\vec{v} + w\vec{w}$$

If no transformation is yet applied to the control points, these vertices of the shark are identical to the original vertices. So far, the control points have not been touched in order to maintain its S,T,U vector, however once the steps above are performed, the program needs to determine the position of the 64 control points (4x4x4) once again so that they are all relative to the U,V,W vectors (in this case S,T,U) in case those get transformed. This step is required in order to preserve the volume of the box, in turn preserving the volume of the shark.

$$\mathbf{F}_{i,j,k}^{pp} = \mathbf{O} + \frac{i}{l}\vec{u} + \frac{j}{m}\vec{v} + \frac{k}{n}\vec{w}$$

The formula above uses i,j,k as indices of the control points where 0 < i,j,k < l,m,n (4x4x4)

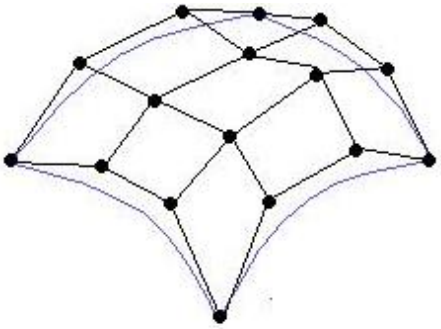
For every global transformation on the control points, $\mathbf{F}_{i,j,k}^{pp}$ (coordinates of the control points) will be recalculated thus after every frame, the function which calculating this gets call after the call the transformation. The program will, based on all the values acquired above, transform all 2560 vertices on the shark using the *trivariate Bézier function* to accommodate the changes in the position of the control points. In another word, the program will iterate through the whole set of shark's coordinates $\mathbf{P}$ and reconstruct them using the formula below…

$$\mathbf{F}(u,v,w) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n}\mathbf{F}_{i,j,k}B_i^l(u)B_j^m(v)B_k^n(w)$$

The formula utilize the *Bézier curve* $B_k^n(w)$ to obtain the deformation of $\mathbf{P}$. The amount of deformation to be applied to the shark is represented by $\mathbf{F}(u,v,w)$ thus the new position $\mathbf{P}$ of the shark is the sum of itself and the difference between $\mathbf{F}(u,v,w)$ and $\mathbf{P}$. The formula to calculate the *Bézier curve* uses iteration on each point of this curve to create a new formation (in the case of 2D line).

$$Q(u) = \sum_{i=0}^{n} p_i B_{i,n}(u) \quad \longrightarrow \quad B_{i,n}(u) = {}^nC_i u^i (1-u)^{n-i} \quad \longrightarrow \quad {}^nC_i = \frac{n!}{i!(n-i)!}$$

However the shark is in form of a 3D model thus it uses the Tricubic Bezier Hyperpatch form of the Bézier *curve.* Once above calculation done, the program render the shark with the new value of $\mathbf{P}$, and this will have the affect of transforming the shark as the control points lying on the cubic frame are moving.

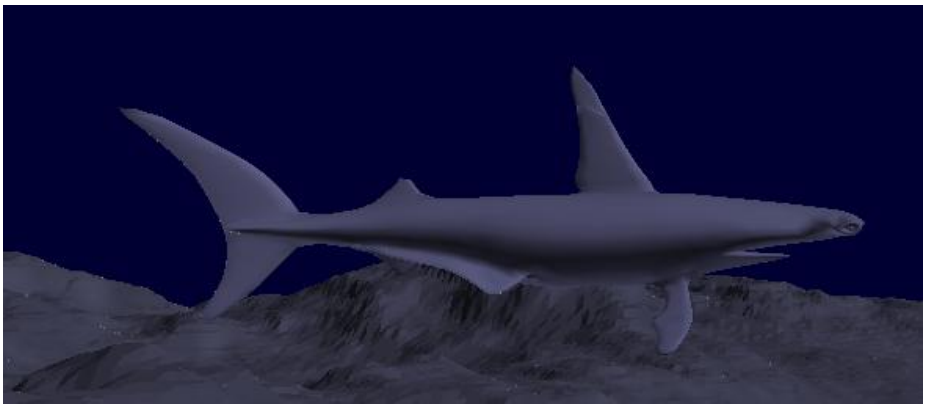


$$\sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} p_{ijk} B_i(u) B_j(v) B_k(w)$$

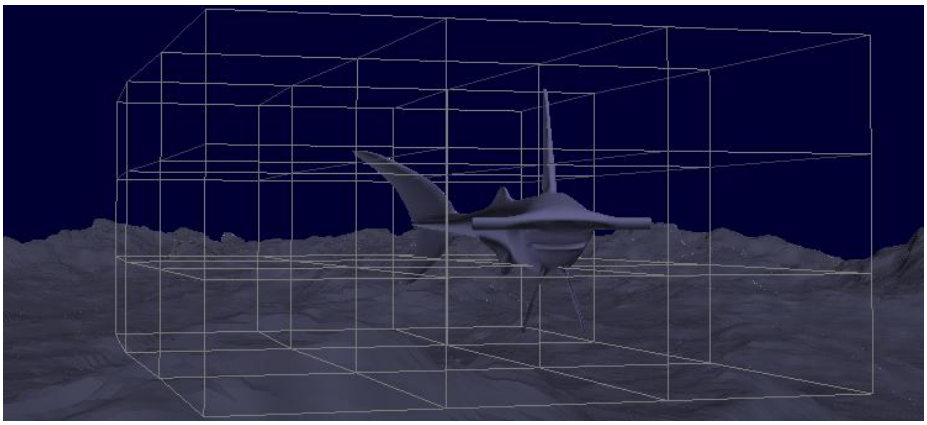
Tricubic Bezier Patch

Result:

The 3D shark model implemented using Free Form Deformation is swimming with the volume preserving motion, and also with highly textured terrain.



The main transformation is applied on the control points located at each intersection of the grid on the cubic frame.



Online References

Sederberg, T.W. and S.R. Parry, ``Free-form deformation of solid geometric models," *Proceedings of SIGGRAPH '86* (Dallas, Tex., Aug. 18-22, 1986). In *Computer Graphics*, *20*, 4 (Aug. 1986), 151-160.

Stony Brook Unniversity, Department of Computer Science, “Deformation”. Date of Access: Mar.3.2010 <http://www.cs.sunysb.edu/~qin/courses/geometry/12.pdf>

University of California at Davis , online computer graphics notes, “Free Form Deformation”. Mar.3.2011 <http://idav.ucdavis.edu/education/GraphicsNotes/Free-Form-Deformations/Free-Form-Deformations.html>

Shimer, Carl, “Advance Topics in Computer Graphics”. Date of Access: Mar.3.2011 <http://web.cs.wpi.edu/~matt/courses/cs563/talks/freeform/free_form.html>