INNOVATION PROJECT REPORT

By Karl Hansson BACVA3 2006
Bournemouth University, UK

MORPHING OF DIFFERENT TOPOLOGIES

Introduction and Background

3D computer animation is today more like puppetry than drawn animation, because it is not as easy to make as good deformations in 3D as it is on papers. This makes things like cartoon animation, where extreme deformations are often required, more difficult in 3D. The problem is that 3D models have a very rigid surface structure, which is defined by the amount of points it consists of and the point connectivity, i.e. which polygon is connected to which point in order to make up the polygons of the surface. It is very difficult to change the pointcount or the point connectivity in mid animation and especially difficult is it to do it seamlessly, since it often results in flickering or snapping motion. In addition, the points are indexed so that each point have a unique identifier which is an integer ranging between 0 and pointcount – 1, this id is used to pair the point with various features like colour and bone influence, etc, inserting points or deleting points may cause the id of other points to change and thus distort any mapping of, for example colour. Point id also plays an important role when editing the model. Morphing the model in an animation requires you to either modify the current mesh or to replace the model.

One can think of drawn animation as using the model replacement technique, for each new frame there is a new sheet of paper. Replacing the model in 3D animation is usually not an option since it requires a lot of very similar models and a way to switch between these shapes to achieve smooth motion which is not very interactive. It is more common to modify or deform a single model to achieve animation. However, modifying the geometry has its limits as well since the topology needs to remain the same. Topology in this context is how the surface is built up, how many points it consists of and how the polygon edges are connected.

Morphing between two very different shapes and being confined to a static topology description is often unsatisfactory since the surface will have too little detail in certain areas to define both shapes equally well and too much detail in other areas where it is not needed for both models. The topology usually is defined parallel with the shape and the polygon edge flow and point density is usually unique to the shape.

The Project Experiment

The idea with this project is to investigate ways in which one could change the topology of an object in mid animation and morphing between two shapes with different pointcount and point connectivity.
The decision to tackle this problem comes from the urge to do things like morphing between shapes with few or no commonalities in their topology, like morphing a human model into a model of a car, or to simply have things growing out of an object, for example a character who grows wings on his back.

Concept

The basic idea was to have the different shapes being defined by hierarchies of deformers and to have the surface and its topology automatically conforming to the deformers, thus breaking the problem into two sub problems; deformation and adaptive subdivision. Having the shape defined by deformers means that you could morph the influence of the different deformer sets and not the different point positions, eliminating the need for the two objects to have identical pointcounts. The surface would then act upon the deformers and subdivide local areas that have been deformed a lot.

The system developed in this paper has been given the name Active Adaptive Surfaces or AAS.

Implementation

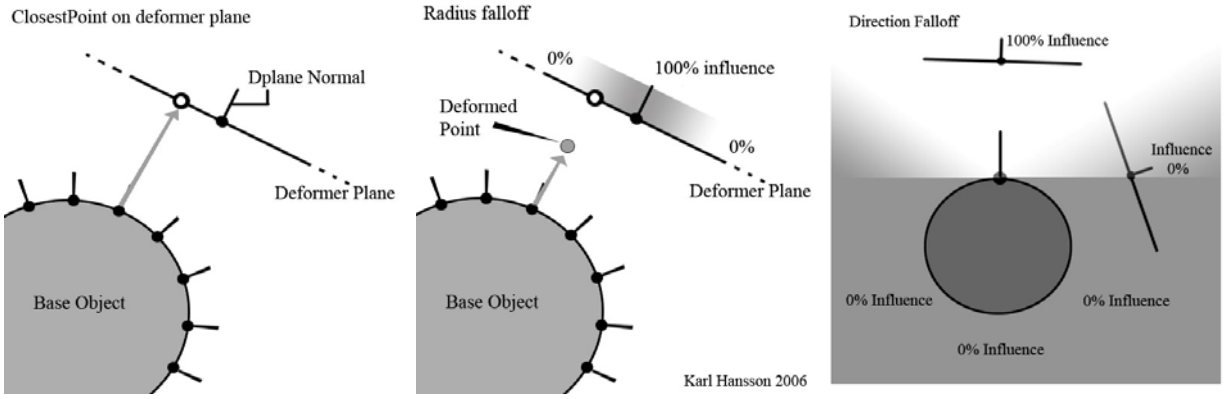
This project utilizes Houdini 8, which has a procedural approach to 3D, and is very suitable for experimentations. All code is written in the VEX scripting language which is native to Houdini.

The Deformer

The first requirement of the deformer is that it needs to be as light as possible in terms of calculation time because a shape could potentially be made up of hundreds of deformers and each one of them needs to be evaluated at each frame. The second requirement is that the deformer has to operate on points but without having to looking at their indices. The deformer also needs to be able to subdivide the polygons it influences.

The first approach was to make a straight forward deformer which translates a set of points in object space. The deformer would be represented by a plane which pulls the points towards or pushes the points away from it self. For the sake of the second requirement, the deformer could not use the indices of the points; it had to use some other way to select the points which it would then operate on. The optional way to pick the points without using the indices would be to select the points which fulfill some requirement, such as certain location or a certain colour or normal, etc. It made sense at the time to use the normals of the object since they would update after each deformation. The idea was to sample the normal of the deformer plane and the normal of each point on the surface and then using the dot product between these two normal vectors to get a deformation scale value to apply to the deformation. This value would then be clamped between 0 and 1 so that points with normals more than 90 degrees away from the normal of the deformer plane would get a deformation scale value of 0 which would not be deformed at all, i.e. points with normals ranging from perpendicular to antiparallel relative to the deformer plane normal would not get deformed at all. This worked quite well as long as the base geometry were convex, however, after several deformers had been applied and parts of the geometry were becoming concave the deformer were no longer able to pick the right points to deform. Also flat parts of the geometry would get deformed uniformly. The problem was that the deformer still evaluated all the points even though just a small set of points was meant to be deformed. The deformer needed improvements to the falloff system so that it would not simply deform points with the right normals but also to take into consideration other things like closeness and direction to the deformer plane center point from the surface point.

The improved version of this deformer would calculate the position of the closest point on the deformer plane from the surface point, by taking the normalized deformer plane normal and dotting it together with the direction vector between the surface point and the deformer plane center point and then adding the current surface point position. The closest point on the deformer plane is the position where the surface point would end up if it had a deformation scale value of 1 (one). The distance from the deformer plane center point to a user defined radial threshold is used as an interpolation reference to calculate the radial falloff. The farther away from the deformer plane center point the lesser the influence of the deformer. The next improvement was to apply a direction falloff by taking the dot product between the direction vector between the undeformed surface point and the deformer plane center and the normal vector of the surface. In essence this means that if the direction to the deformer plane center point is at an angle more than or equal to 90 degrees away from the surface point normal than the deformer has no effect on that surface point. These improvements made the point picking better but not perfect, there were still problems in folded areas.

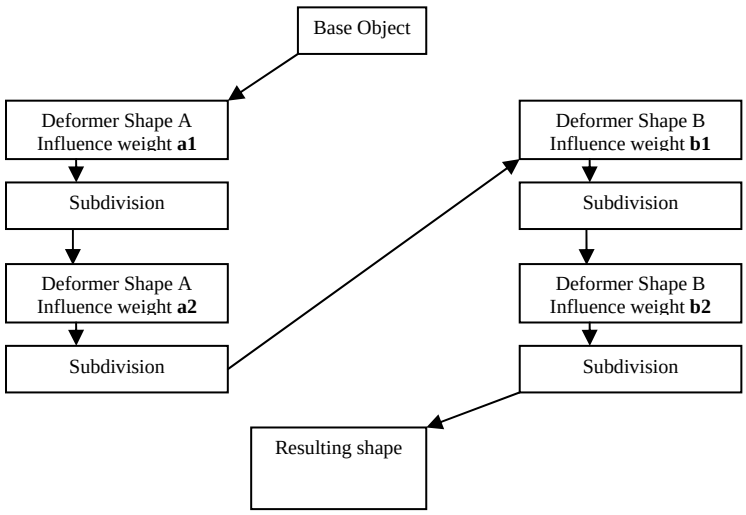


The deformer uses the vector, between the surface point and its closest point on the deformer plane, to determine the direction of the displacement. The closest point on the deformer plane is also used to give the point a radius falloff, the further away from the center of the plane the point is the less influence the deformer will have on that point.

The first attempt at local subdivision was simply to group the points that had been deformed more than a certain threshold and after they had been deformed the polygons connected to those points would get subdivided one time. It proved, however, to be very calculation heavy and after adding just a few deformers with subdivision to the shape Houdini would crash. This put an awkward limit on the experiment, because one could not model very much with just a few deformers.

The Idea at this stage would be to have a main influence weight on each deformer and by scaling these weights between 0 and 1 for each deformer the shape would change and with it the topology as well. This is quite good approach because you can animate the scale factor of different deformers at a different speed, which makes the shape blending more visually interesting. One could also animate the position and orientation of the deformers which would make the blending appear less linear. But there is practical problems with this idea as well, since there can potentially be hundreds of deformer making up one shape it would take a lot of work to set up a blend. Also editing the blend, like shifting it in time or change the pace of the blend would get very tedious. One would need some kind of interface to drive the blending.

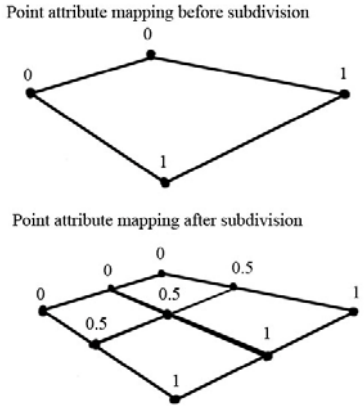
Another problem at this stage was that all the deformer nodes for both shapes were connected in a chain one after another. This made it difficult to maintain the deformer nodes and to identify which nodes belonged to which shape. More ideal would be to have each shape as a separate branch of the hierarchy and then connect all of the branches into a single blending node. But at the time this did not make much sense, because each branch would then produce separate objects, with different topology, and thus stepping away from the idea to blend between the deformers and not point positions, and one would again have to take into consideration the indices of the surface points. The deformers would in that case be made redundant because the objects could then just as well be modeled using traditional tools and the blending node would be more like a regular blendshape node that would require each object to have the same pointcount. Since the goal of this project is to find ways to morph between shapes that have different pointcounts and different connectivity, the feature of having the different branches connect up to one blending node seemed impossible.



The first version of the system looked like this. It's a very basic approach which has just one long chain of deformer nodes and not separate branches for the different shapes. The blending was achieved by scaling the influence weights between 0 and 1. For example, to show shape A one would set the weights **a1** and **a2** to 1 and **b1** and **b2** to 0. To show shape B one would set the weights **a1** and **a2** to 0 and **b1** and **b2** to 1. And to blend them one would interpolate both shapes weights so that their sum adds up to 1.

The Blender

One feature of polygon surfaces is that one can map attributes to the points on the surface, these attributes would typically be things like colour, texture, bone weights, etc. If an attribute is mapped out on a polygon surface and if the surface is then subdivided, the new points generated from the subdivision will also have the attribute and the value of the attribute for these new points will fit into the mapping. For example, two points sharing a polygon edge have attribute *w=0* and *w=1* respectively, if then the edge is divided into two edges; there will be a new point in between the two old points. This new point will have the attribute value *w=0.5*, i.e. the average of the two old points. Houdini makes it possible for the user to make custom point attributes; a feature which was very useful for taking the project to the next level.



A demonstration of how point attributes are mapped after subdivision. The new points on the surface will get an average attribute value relative to the surrounding points.

The way a blendshape function usually works, is that it moves the points of the geometry towards the location of the same point, with the same index number, in another shape. That is why these shapes usually need to have the same number of points. It is a very difficult to blend two shapes with different topology and there is no real good solution today.

What if one could store the deformation as an attribute on each point without actually deforming the surface at the deformer? The idea might seem strange but it makes sense when it comes to blending

the deformers. It is basically a matter of changing the order in which things are happens. The deformer would still have the role of deforming and defining the geometry but on a more indirect level. Since the two shapes comes from the same base geometry and the deformers haven't physically changed or added any points to the geometry before the blender, the two shapes will have the same pointcount and the same connectivity, but the two shapes will carry different deformation descriptions in their points attributes. The blender would now have a slightly different role; it will not blend the point positions of the two shapes as is normal, but rather blend the deformation attributes.

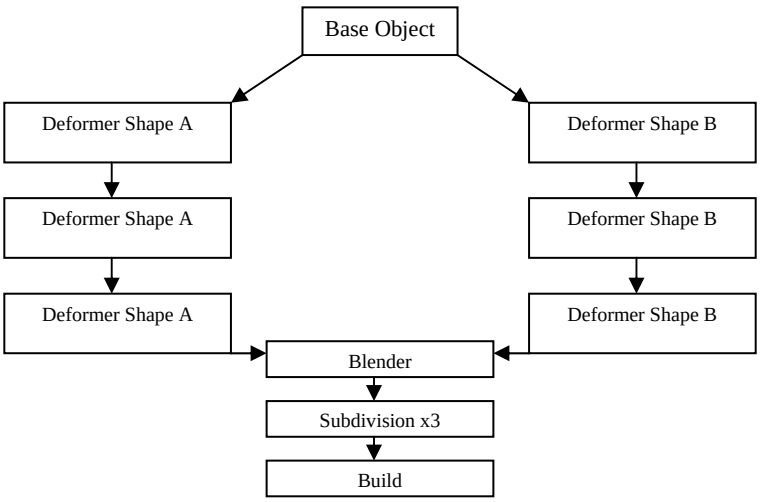
The next step was to solve the subdivision problem, since the idea of this improvement was to have the blender work on separate shapes branches of the hierarchy, there could not be any subdivision before the blender so it have to happen after. The question now was; where on the geometry should the subdivision be applied? One could group points, in the blender node, that has been deformed more than a certain threshold and subdivide the polygons in direct connection with those points. Developing on this idea, the decision of which points to add to the subdivision group would be done in the deformer rather than in the blender. This also made it possible to have more than one subdivision group in order to have several iterations of subdivisions, which is useful when some areas are deformed so much that only one subdivision level would not be enough to maintain the level of detail. Another point attribute was then added which stores an integer number that reflects the number of times the polygons sharing that point would get subdivided. Each deformer checks whether a point is deformed more than a user specified threshold and if so; the subdivision attribute is incremented by one.

In Houdini there are some restrictions where one cannot, in a subdivision node, change division level on per face basis, for example, it is not possible, in a single subdivision node, to subdivide one face to the level 1 and another face to the level 3, for this you would need to have two subdivision nodes. In order not to get too many subdivision nodes, for the sake of performance, a maximum limit of subdivision levels was set to 3, which was fine for this project, but is probably not enough for regular content creation.

In the blender node, after the blending had taken place, the points which are to be subdivided are put into three groups according to their subdivision attribute, i.e. points with a subdivision value of 1 is put in group 1 and points with subdivision value of 2 is put in group 2 and so on. These point groups are then converted to polygon groups and fed into one of three subdivision nodes following the blender node. Because the subdivision needs to happen after the blending node but before the physical deformation, a separate build node needs to be created to handle the deformation.

The build node

The final node handles the physical deformation; it is at this point the base object is turned into the shape that is the result of all the deformers and the blending node. This node simply displaces each point according to its deformation attribute and because of the automatic attribute mapping, the new points, generated from the subdivision, also has a deformation attribute and they are displaced as well.



The last version of the system is more advanced. The two shapes are defined in separate branches which feed into the blending node. Note that there are no subdivisions happening before the blender which means that the two shapes will have the same pointcount. When the two shapes have been blended the geometry will be sent to the subdivision where the appropriate faces will be subdivided. Finally the build node will displace the points resulting in the final shape.

Further improvements to the deformer

The system was working quite well; it morphed between to different shape which effectively, in the end, had different pointcount, but there were also problems with the way the deformers worked. The deformers used the normals of the base object to select the points to operate on and the deformation of a point could only happen in a direction no more than 89 degrees angle from the normal, and at that an angle the deformation would be minimal because of the deformation falloff. This meant that the point of reference were always the same and direction of the deformation was also limited to roughly the same direction, which in turn mean that you could only model things coming straight out of the base object or going into the base object. Curved shapes were almost impossible to model. It was clear that some other way of selecting and deforming points were needed.

Sphere of influence

Instead of using the normals of the base object as a reference, it made sense to use some kind of influence object. The influence object is just a point in space that is represented by a sphere and the radius of the sphere represents the falloff range of influence. Any surface point outside the sphere would not be influenced and any point inside the sphere would have a weighted influence depending on how far from the center of the sphere it is. This is a more interactive way of controlling which points to operate on and it also means that you can work on the deformation data directly and not on the physical point of the base object. For example, a point on the base object has the location **a** = [a_x, a_y, a_z], and the deformation attribute **b** =[b_x ,b_y ,b_z] which may be some place different in the 3D space, the influence object have to be placed with respect to the deformation **b** and not **a**. In Houdini you can edit the deformers early on in the hierarchy and at the same time se what the final output looks like, this makes it very easy and interactive to place the influence object in the right position. This also means that the deformer are now using the influence object as reference and not the normal and there is no longer a limit in which direction the deformation can be applied and as a result there is no longer a problem to make bendy shapes.

Analyzing the outcome of this project

To visualize this whole system using a metaphor of a house building process, one can think of the different deformer branches in the shape hierarchy, as teams of architects that defines the different shapes, they only describe the shape and how much material to use as blueprints. The blender can be thought of as the house buyer who picks features from the different architect teams. And the final

build node can be thought of being the construction firm delivering the final house.
It is quite hard to model anything recognizable using this system. Part of the problem is that the deformers are very basic and can be improved further with better control of falloff. The major problem, however, with this system is the subdivision, optimally the subdivision should happen at the deformer in order to keep sufficient detail to model with, for performance reasons as well as for the sake of the blending this is not possible at this stage, it is an area which deserves more research. The fact that the subdivision happens after all of the deformation can result in some very ugly artifacts.

Other Solutions

As was said before there are no real good solution to morphing between shapes with different topologies, but there are some interesting research going on in this area.
One of the most interesting and perhaps controversial ideas is that of point surfaces. A big hurdle to overcome is that the mesh geometry needs to maintain some kind of connectivity between the points, adding new points you can do with subdivision but to change the connectivity between the points to get the optimal topology description is quite hard and very inefficient. One solution to this problem would be to not having any mesh to worry about. The very interesting siggraph paper, “Shape Modeling with Point-Sampled Geometry“ [Mark Pauly, Richard Keiser, Leif P. Kobbelt, Markus Gross] describes the implementation and use of point surfaces. Point surfaces are quite simply surfaces that are made up of points but without a mesh which is more commonly related to the areas of 3D scanning. Each point is the center of a circle primitive which in the surface slightly overlaps each other in order to give the impression of being a continuous surface. The lack of mesh is a unique feature that would suit the areas of shape morphing very well. In their paper, Shape Modeling with Point-Sampled Geometry, Mark Pauly and his colleagues describe a technique where points are scattered over a regular polygon of parametric NURBS surface, the points then uses information of the underlying surface to map out normals, colour and texture etc. The point surface can then be further deformed using various sculpting tools, and the surface is then maintained using a moving least square algorithm. In essence this becomes like a super elastic skin which inserts or deletes point primitives in order to maintain the surface density. The drawback to this system is that the surface appears slightly grainy or blurry when rendered, but that might be subject to future render enhancements.
Another way to blend between two different shapes might be to use metaball surfaces, it’s a technique that has been around for a long time and is probably the best way of dealing with shape shifting. It is, however, not very easy to model with metaballs they have, especially when it comes to model fine detail.

Conclusion

Morphing shapes with different topologies is a very complex problem, it involves not only changing the shape of the geometry but the topology also needs to change. Mesh topologies are very rigid and because there can be an infinite number of different meshes and the requirement of each mesh might be different, one cannot easily construct an algorithm that changes the topology for any mesh. This project explores the idea of having the surface subdividing it self to meet the requirements of different sets of deformer. It is an idea of having the surface topology conforming to the modeling tools, and to have the modeling tools defining the shape rather than to have point positions defining the shape. The surface system developed in this project does not, however, change the point connections, it merely uses existing subdivision tools to add extra detail where needed. This is not optimal since it makes a very ugly topology which deforms quite badly, but for this experiment it was good enough to show that the shape blending system works. With the shape and the topology being defined by the modeling tools, i.e. the deformers, one can scale the influence of different deformers to change the topology as well as the shape.

References:

Alan H. Barr
Global and local deformations of solid primitives
ACM 0-89791-138-5/84/007/0021
1984
Computer Science Department
California Institute of Technology
Pasadena California

Mark Pauly, Richard Keiser, Leif P Kobbelt, Markus Gross
Shape Modeling with Point-Sampled Geometry
ACM 0730-0301/03/0700-0641
2003

Anders Adamson, Marc Alexa
Anisotropic Point Set Surfaces
ACM 1-59593-288-7/06/0001
2006
Department of Computer Science
TU Darmstadt

Appendix A:

The accompanying movie files found in the movies folder:

AAS001.avi	Shows the first successful blend between one shape defined with three deformers and another shape defined by six deformers. It also shows what happens when the base object is rotated before the deformers.
AAS002.avi	The same blend as in AAS001, here displaying pointcount.
AAS003.avi	Shows the first successful test which results in a bended shape using the sphere of influence.
AAS004.avi	Shows the first attempt to model something recognizable with this system. This is a nose.
AAS005.avi	Shows the second attempt to model something recognizable. A shape blend between a human head and a eagle head. Pointcount displayed
AAS006.avi	Shows the final attempt. A much more detailed head and a eagle head is blended. Also shows what happens if the deformers are moved around in order to get non linearity. Pointcount is shown.
Metaball.avi	A metaball model was created to se how easy it is to model with metaballs.

Appendix B:

Vex code files:

aas.vex	The latest vex code for the blender node
aasGD2.vex	The latest vex code for the deformer node.
aasBuild.vex	The latest vex code for the build node.

Scene files:

The latest and most complete scene file is the AAS007.hip**.**

Otl files:

The otl file AAS2.otl holds the latest aas operators for Houdini.