VDB Deformer: VDB Free-form Deformation in SideFX Houdini

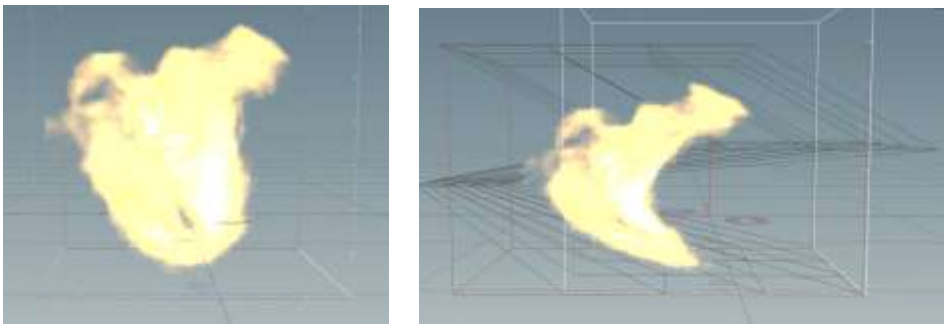

Figure 1: Left is input flame simulation, right is resulting deformation

John Noonan

1 Abstract

VDB Deformer is a tool built into SideFX Houdini as a Surface Operator (SOP). This tool allows for free-form deformation (FFD) of OpenVDB primitives through the use of a lattice. By manipulating the shape of a volume the user is allowed greater control of the work and facilitates faster iteration.

2 Motivation

Volumes are used frequently in computer graphics, particularly for animated elements in smoke, fluid, and fracture effects. These effects are often created through complex simulations which may require lengthy compute times. For uses such as feature animation these simulations may be required to hit a certain visual mark as defined by directors which can be difficult to create through adjusting simulation parameters alone. Performing iterative simulations may also be too costly and still does not guarantee the desired effect. As such it is often necessary to edit an existing simulation or volume representation rather than attempt to create a new one. Art-directed volume deformation offers a way to work around these

limitations. Performing user controlled edits on simulations offers additional control. Example uses may be: editing a static volume, creating low cost collision by having the simulated volumes closely avoid other objects, sculpting a very specific silhouette, or allowing for a physically correct simulation to selectively perform an unrealistic action.

3 Related Work

Free-form deformation was made ubiquitous in computer graphics by Sederberg and Parry which introduced a method for “sculpturing solid models” [Sederberg 1986]. FFD is a versatile tool which allows for indirect modification of geometry by superimposing a caged lattice around the model which may be edited. The FFD is defined in terms of the trivariate Bernstein polynomial. The lattice serves as a local coordinate system which may be manipulated by the user. Each vertex of the model may be mapped to the modified grid which relocates the point in global space. For the most part this form of model deformation is used on discrete polygonal meshes or implicit surfaces however extensions to volumetric data exist. Ecker et. al used a volumetric deformation technique which involved sampling density values to points centered on each voxel, deforming the resulting point cloud using standard geometric deformation, and then transferring the values from the points into the new volume [Ecker 2016]. Wrenninge and Rice created rasterized data and a custom lattice primitive which makes it possible to deform existing volumes with intuitive control used in the 3D software SideFX Houdini [Wrenninge 2016].

4 Technique

4.1 Utilized Technologies

VDB Deformer leverages several existing frameworks. The tool is built in as SOP node in SideFX Houdini by using the Houdini Development Kit, a C++ API that allows for plugin

development. Additionally the program uses OpenVDB, an open source hierarchical data structure for the efficient representation of sparse volumetric data on a 3D grid [Museth 2013]. The suite can represent both density fields and signed distance fields, both of which are capable of being used with the deformer.

4.2 Data Structures

This project benefits from leveraging powerful existing APIs, but within these tools two data structures were created. The class SOP_Volume_Deform is the implementation of the node used in Houdini. It defines the parameters and required inputs as well as performs all calculations in the software. This process is called “cooking” and is launched by the function “cookMySop”. SOP_Volume_Deform has no local variables as the parameters it edits must be recalculated each time a parameter in the node tree is edited. Thus inputs are queried inside cookMySop and used to edit the global detail, an inherited value that Houdini will use to output the result of our node.

The other data structure created is “MyLattice”, an implementation of that described in Sederberg. This class stores the input rest and deformed lattices as geometric details. From these it calculates the local axis S, T, and U and the origin p0. Using linear algebra given any world position it can calculate the local coordinate position (s,t,u) such that if the point is inside the rest lattice each component will be between 0 and 1. All deformation work is handled by this class.

4.3 Data Gaps

As stated earlier, FFD is primarily concerned with the deformation of polygonal data. A distinction between polygonal and volumetric data is that polygonal data is defined by connected vertices while volumetric data is (for the most part) comprised of discrete grids. If a vertex of a square is dragged outward, the quadrilateral is still defined continuously by the set of vertices and edges. If a voxel is moved in a 3D grid, there is no direct connection that guarantees the grid will still be continuous (see figure 2 for an example). As such a major concern with volumetric deformation is maintaining the initial coherence of the data set without introducing gaps. This can occur when the lattice represents a stretched shape where new cells will be activated and empty cells may exist between the new deformed positions. Unfortunately, merely rasterizing the voxels as points and then performing deformation will lead to discontinuities and require a high resolution grid in order to not show these errors.

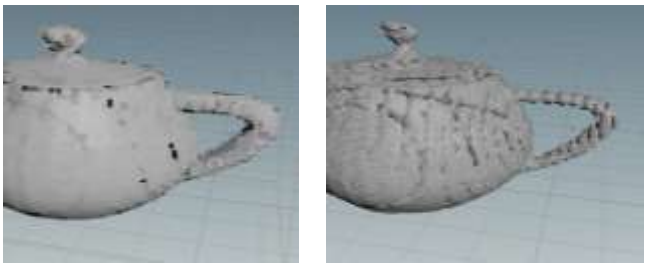

Figure 2: Example of a volume with gaps on left and the polygonalization of that volume

4.4 Algorithm

A goal of this project was to try and deform the volumes without direct rasterization to a point cloud. As such the basis of the algorithm is for every input voxel calculate the deformed position in world space and then write the value of the voxel into a new volume at the deformed position. However, this forward deformation will result in gaps. Therefore, the proper algorithm would be for each cell in the destination grid that is inside the deformed lattice, determine the local coordinate of the voxel and then sample the original grid at that world space position corresponding to the same local coordinate and interpolate any value that would exist at a fractional grid index. This is of course the backwards process the typical FFD. The main

challenge with this approach is that for any deformed world space position no local coordinate value is known. No previous work has been found exploring this topic.

A “reverse deformation” seems to be the solution, however this is not very feasible as it would require a complex computation of the inverse of Bezier interpolation and is unlikely to have one result. An attempted solution was to interpolate the local coordinates by performing trilinear interpolation of the deformed point inside the subdivision cell of the deformed lattice. This form of reverse deformation is currently still being worked on. Therefore the only deformation available at the moment is the forward FFD which regrettably produces grid holes.

4.5 Technical Challenges

Throughout the development of this project there were numerous challenges. One significant setback was difficulty compiling with the Houdini HDK and OpenVDB. Proper file inclusions were not always obvious and throughout there was difficulty building with the compiled libraries. An example of this is how the OpenVDB API uses the file openvdb.lib while the HDK uses openvdb_sesi.lib located elsewhere. Because of these difficulties the project is currently compiled using the Houdini command line tool “hcustom” with explicit file inclusion. Unfortunately, building with cmake is currently not working. Once the project could be compiled, the next hurdle to overcome became the difficult task of learning the HDK. It is quite complex and at times very unintuitive. One problem was trying to edit the input VDB as it requires multiple dynamic casts to convert to the correct data type. Additionally as Houdini is a proprietary program, a lot of the inner workings are hidden and the auto created documentation is not always helpful, such as there being no explained distinction between the functions xsize() and sizeX(). Another challenge was iterating through voxels. There was some confusion as to the coordinate space of the VDB grid as each voxel has “index space” coordinates that defines the three dimensional grid and a “world space” coordinate. It can be confusing which way to

access a voxel is the correct way to be used in certain situations, which led to a lot of trial and error.

5 Results

5.1 Usage

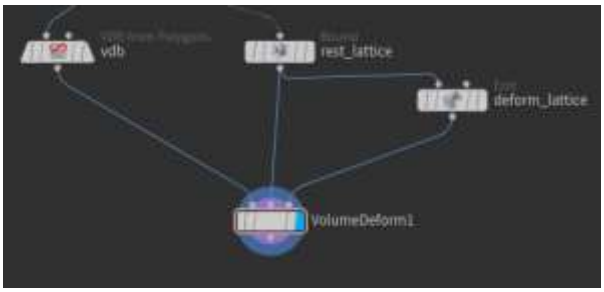

Figure 3: Example node layout

Users create a VDB representation natively in Houdini and a uniform parallelepiped to represent the lattice. While any axis aligned uniformly subdivided box will work it is recommended to use a bounding box of the volume for results to be the most intuitive. This bounding box should be of the largest size required to contain the volume over the entire animated timeline. A deformed copy is created of the rest lattice which may be edited in real time by dragging or programmatically editing the position of the cage control points. The VDB, rest, and deform lattice nodes are connected to the deformer node and the xyz divisions of the lattice are specified as parameters (see figure 3).

Examples

VDB Deformer has been tested on several use cases. The most basic is the static deformation of a volume (figure 5). More complex than this is a volume with an animated lattice (figure 4). Finally a test was created on an animated flame simulation (figure 1). As is shown, the deformations look plausible and usage was intuitive. Save for the gaps in the data, the results are pretty good.

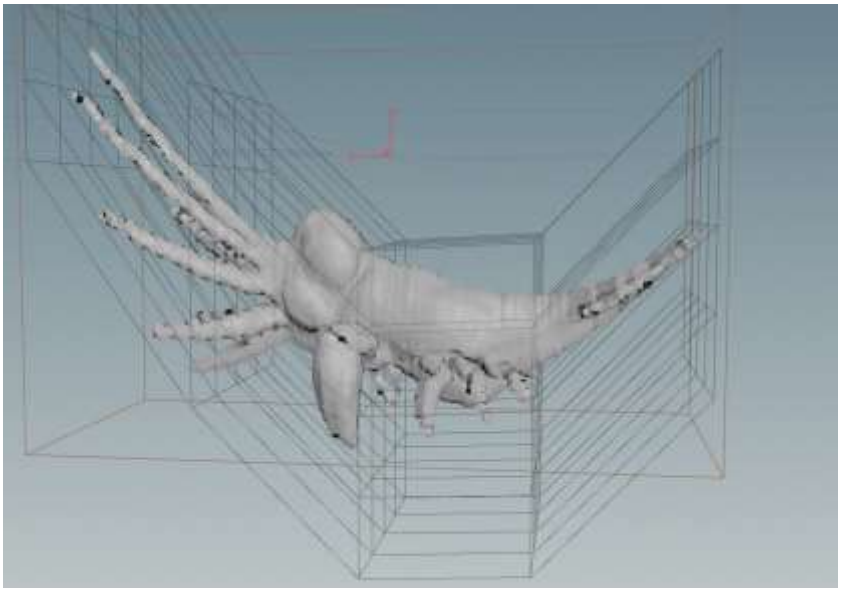

Figure 5: Deformed volume with animated lattice control points along a sine wave.

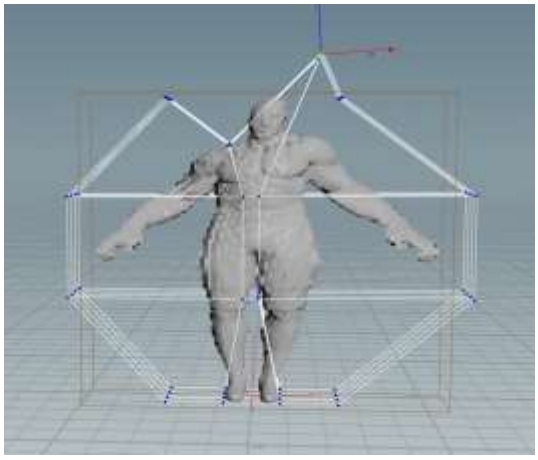

Figure 4: Deformed static volume with high compression and stretching

Future Work

There are several parts of the project that could stand to be improved upon. Most obviously is the continued implementation of the “reverse deformation” in the attempt to remove data gaps. Additionally the deformer currently only works on VDBs with floating point data. As VDBs can represent arbitrary datasets it would be nice to template this tool to be able to use any input type such as vectors, integers, and booleans. Furthermore the deformer can only operate on one grid. In Houdini it is not an infrequent occurrence for an object to be represented by multiple VDBs (for example an explosion may have a VDB for the smoke and fire as well as other data such as velocity and temperature). In order to properly deform the entire collection of volumes multiple deformations would need to be executed, which is inconvenient and more costly.

Work

This project was worked on by John Noonan with the assistance of both the ODFORCE Houdini Development Kit and OpenVDB forums. It took approximately 80 hours to get to this point.

References

Thomas W. Sederberg and Scott R. Parry. 1986. Free-form deformation of solid geometric models. In Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH '86), David C. Evans and Russell J. Athay (Eds.). ACM, New York, NY, USA, 151-160. DOI=http://dx.doi.org/10.1145/15922.15903

Gregory Ecker, Ravindra Dwivedi, and Ilan Gabai. 2016. Directed volcano getting the most out of your simulations. In ACM SIGGRAPH 2016 Talks (SIGGRAPH '16). ACM, New York, NY, USA, Article 3, 1 pages. DOI: https://doi-org/10.1145/2897839.2927454

Magnus Wrenninge and Michael Rice. 2016. Volume modeling techniques in The Good Dinosaur. In ACM SIGGRAPH 2016 Talks (SIGGRAPH '16). ACM, New York, NY, USA, Article 63, 1 pages. DOI: https://doi.org/10.1145/2897839.2927397

Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. ACM Trans. Graph. 32, 3, Article 27 (July 2013), 22 pages. DOI: https://doi-org/10.1145/2487228.2487235