An Object-Level Destruction Tool in Houdini
Sewang Kim

Submitted in Partial Fulfillment of the Requirements
For the Degree of Master of Fine Arts in Visual Effects
at
The Savannah College of Art & Design
© January 2013, Sewang Kim

The author hereby grants SCAD permission to reproduce and to distribute publicly paper and electronic thesis copies of document in whole or in part in any medium now known or hereafter created.

Signature of Author and Date: __________________________ / ______ / ______

________________________________________ / ______ / ______
Deborah R. Fowler (Sign here) (Date here)
Committee Chair

________________________________________ / ______ / ______
Malcolm A. Kesson (Sign here) (Date here)
Committee Member

________________________________________ / ______ / ______
Stuart Robertson (Sign here) (Date here)
Committee Member
An Object-Level Destruction Tool in Houdini

A Thesis Submitted to the Faculty of the Visual Effects Department
in Partial Fulfillment of the Requirements for the
Degree of Master of Fine Arts in Visual Effects
Savannah College of Art and Design

By
Sewang Kim

Savannah, Georgia
January, 2013
## Table of Contents

List of Figures ................................................. 1  
Abstract ......................................................... 2  

1. Introduction .................................................. 3  
   1.1 Problem with Houdini’s Workflow .......................... 3  
   1.2 Solution / Purpose .......................................... 4  

2. Previous Work .................................................. 6  

3. Technical Discussions ......................................... 8  
   3.1 HDA / Python Module ....................................... 8  
   3.2 Voronoi Fracture Surface Node .............................. 9  
   3.3 Bullet Solver ................................................ 12  

4. Methodology .................................................... 14  
   4.1 Object-Level Shattering ..................................... 14  
   4.2 Automatic Dynamics Settings ................................. 16  
   4.3 Customizable Particle System ............................... 18  

5. Conclusions ..................................................... 21  
   5.1 Future Work .................................................. 21  
   5.2 Conclusions .................................................. 21  

User Guidelines ............................................... 23  
Bibliography ...................................................... 26
List of Figures

1.1 Shattering in two different contexts ............................................. 4
1.2 The destruction tool written in Python ........................................... 5
2.1 A scene from 2012, visual effects by Digital Domain .......................... 6
2.2 A Voronoi diagram ........................................................................ 7
3.1 Different operator types ................................................................. 8
3.2 A built-in “Voronoi Fracture” node .................................................. 9
3.3 Inputs & Outputs of the “Voronoi Fracture” node .............................. 10
3.4 Two useful primitive groups ............................................................ 10
3.5 “Piece Group Prefix” parameter ..................................................... 11
3.6 Two different ways to group primitives .......................................... 11
3.7 “Primitive Cell Point” parameter ..................................................... 12
3.8 Overlapping collision boundaries between pieces .............................. 13
4.1 Primitive groups ............................................................................ 14
4.2 The object-level shattering workflow .......................................... 15
4.3 A custom attribute “shatter_level” .................................................. 16
4.4 Make Active & Make Passive buttons ............................................. 16
4.5 A flowchart of how the tool brings fractured pieces into the dynamic level 17
4.6 Interactive dynamic-weights setting ............................................. 18
4.7 An example of applying the particle system .................................. 19
4.8 How the particle system works ..................................................... 20
4.9 Parameters for the control of particles ............................................ 20
An Object-Level Destruction Tool in Houdini

Sewang Kim

January, 2013

Destruction tools in Houdini have been successfully used to create visual effects in production but are time consuming. Creating a destruction tool within Houdini that allows effects artists to shatter multiple buildings quickly and visualize these fractured pieces simulated in a simple way would enhance the current workflow. The tool allows selection of pieces of models on an object level (chunks) rather than component level (points or primitives) to provide the artist an easier, more intuitive way with more immediate visual feedback to deal with the destruction processes, fracturing and dynamics. As a result, this artist friendly tool can help visual effects studios speed up their entire destruction pipelines increasing realism and performance.
1. Introduction

The destruction process today has become a vital element of many important visual effects pipelines due to high demand in the film industry. Accordingly, many major studios, such as Digital Domain, Industrial Light & Magic, Double Negative, and Sony Pictures Imageworks, have already developed their own destruction pipelines for shots where very heavy amounts of Rigid Body Dynamics (RBD) data is being dealt with by technical directors, software developers, or effects artists in production. More importantly, those companies have begun replacing the effects software on which they traditionally depended with Side Effects’ Houdini. David Horsley, FX Supervisor at Rhythm & Hues, says “Houdini is flexible and its productivity is from the lack of limitations” [Side FX 1]. Also, Philip Prahl, CG Effects Animation Lead at Digital Domain, states “One of the strongest features of Houdini is how easy it is to navigate and coordinate different contexts such as dynamics and particles” [Side FX 2]. This confirms that the procedural workflow within Houdini is fairly efficient and powerful enough to handle the effects work of a production. Even so, it can be disadvantageous for a studio to approach destruction effects with such procedural workflow in Houdini.

1.1 Problem with Houdini’s Workflow

As a result of Houdini’s procedural nature, when developing destruction effects many effects artists procedurally shatter a 3D model in the geometry context where everything becomes component-based (points or primitives) rather than in the object
context where multiple components become individual objects. When working in the geometry level in Houdini, giving unique group names to each fractured piece is the only way to distinguish one from another (see figure 1.1).

For this reason, it is very difficult for an artist to work with or make changes to a specific shattered chunk. For example, when the director wants only several chunks among all those fractured to be split further into smaller pieces, the artist has to spend considerable time figuring out the group names of those pieces. In addition, when the artist starts working with RBD simulations, things become more complicated. He or she has to tell Houdini whether those chunks are to be affected by the RBD solver or not by specifically adding or deleting their group names in the dynamics context within Houdini.

1.2 Solution / Purpose

The primary purpose of this thesis is to overcome such limitations and provide the artist an easier and more intuitive way to deal with the destruction processes, fracturing
and dynamics. A Houdini Digital Asset (HDA) written in Python, a part of my thesis project, allows users to cut their 3D models in the object-level (see figure 1.2).

Thus, he or she can simply select any chunks on which they want to work, without having to search through a large number of primitive group names (see figure 1.1). Also, this asset simplifies the time-consuming process a user needs to manually assign the group name of each shattered piece to either an active or passive RBD object, by offering simple parameters for the user. Finally, the tool provides a customizable particle system that can be used to create additional details such as debris, dust, or smoke.

Figure 1.2 – The destruction tool written in Python
2. Previous Work

When it comes to building destruction scenes, 2012 [2009] holds the best examples of any movie, as Mike Seymour at FX Guide writes, “there is no doubt that the massive work done for the film 2012 was a turning point for the industry. Until that time nothing had been seen on that scale.”

Although many major studios such as Double Negative and Uncharted Territory participated in the movie, destruction effects done by Digital Domain came into the spotlight [Seymour]. Digital Domain found it necessary to expand their proprietary RBD pipeline for the sequence where buildings and houses collapse, highways and valleys crumble, and huge land masses go under the ocean [Seymour].

Digital Domain used a Voronoi diagram, a mathematical concept used in various fields, for the shattering process [Failes]. The Voronoi diagram refers to dividing a space into areas determined by distances to a specified discrete set of points on the surface [Failes] (see figure 2.2).
One can simply get naturally divided pieces by utilizing the formula. Moreover, these decomposed pieces become all convex shapes, which work perfectly well with Houdini’s Bullet Engine. The studio relied on Bullet, one of the most popular physics libraries, for their RBD simulations [Failes]. As Mohen Leo, visual effects supervisor at Digital Domain, says, “Bullet is very simple, but it’s fast, stable and, because it’s open source, you can build on top of it” [Failes]. This also means: “artists could perform a number of iterations and variations and work more creatively, as opposed to waiting overnight to see if it looked OK”, according to Leo [Failes]. The massive destruction effects of 2012 done by Digital Domain served as a momentum for Bullet to be a primary physics engine for many other visual effects companies [Seymour].
3. Technical Discussions

3.1 HDA / Python Module

A Houdini Digital Asset (HDA) comes in very handy when developing a tool within the Houdini environment. We can simply start building an HDA by either writing scripts that Houdini recognizes, either in the Vector Expression (VEX) language or Python, or encapsulating any necessary nodes into one. Depending on the context in which we would like to use the asset, there is an option where one can specify among different selections such as “Geometry” or “Object” operator (see figure 3.1).

![Different operator types](image)

Figure 3.1 – Different operator types

As discussed earlier, the Geometry Operator lets us deal with geometric components (points and primitives) and the Object Operator allows us to work in an object-by-object environment. A user can access Houdini using Houdini Object Model (HOM), an application programming interface (API) composed of Python modules, functions, and classes [Side FX 3]. The Python module inside the HDA works just like Maya Embedded Language (MEL). For example, the tool automates various processes, like the creation of nodes, setting their parameters, and linking these parameters’ values to the tool so the user can tweak them and see the changes right away. Also, writing Python
scripts is an excellent method of handling string data, which adds, deletes, or replaces any primitive group names if necessary.

3.2 Voronoi Fracture Surface Node

The cutting system of my tool relies on the “Voronoi Fracture” surface node, a built-in HDA inherent to Houdini (see figure 3.2).

![Voronoi Fracture Node](image)

Figure 3.2 – A built-in “Voronoi Fracture” node

I chose this instead of developing my own because the node is already well-developed for splitting a mesh into smaller pieces. Basically, this node needs two inputs, a target mesh one wishes to cut, and points used as Voronoi cells [Side FX 4]. The node then generates a bunch of evenly shaped pieces whose number is the same as the number of points provided as the input. The original mesh is cut multiple times between every two arbitrary points in a direction perpendicular to both (see figure 3.3).
The node not only produces fracturing but also keeps very useful primitive groups named: "inside" (whose newly-created primitives are inside the original mesh), and "outside" (whose primitives are outside of the original surface) (see figure 3.4).

The "Voronoi Fracture" node also has a parameter named "Piece Group Prefix" which allows the user to have a unique group name per chunk created (see figure 3.5).
Although this may be useful in some situations, I faced a serious issue when using this parameter in dynamic simulations; even though two cells in figure 3.6 are given as the input, the use of the parameter causes four separate objects to be created. Among these four pieces, “piece 1” and “piece 3” are overlapping each other. This causes an explosion when converted to RBD objects, a problem that occurs between intersecting parts of a model, unless the modeler has created one with no overlapping parts.

---

**Figure 3.5 – “Piece Group Prefix” parameter**

**Figure 3.6 – Two different ways to group primitives**
Supposing no one would have such a perfect model, I had to find a different way to distinguish one chunk from another. There is another parameter named “Primitive Cell Point” in the “Voronoi Fracture” node (see figure 3.7).

![Figure 3.7 – “Primitive Cell Point” parameter](image)

The parameter allows one to split the model, regardless of whether it has overlapping parts or not, based on the number of cells. Each split piece then remains a single RBD object. This fixes the intersection problem in the figure 3.6 by allowing the offending objects to be considered as two separate pieces, instead of four.

### 3.3 Bullet Solver

Once those fractured chunks are created, the user needs to appropriately bring them into the dynamics context for the RBD simulation. Houdini supports three different types of solver engines for rigid body dynamics. The Open Dynamics Engine (ODE) can handle large numbers of simple objects as it uses simple geometric primitives such as spheres or cubes for collision detection [Side FX 5]. This engine, however, does not work with “RBD Fractured Objects,” which is the key method my tool uses to bring
fractured pieces into the dynamics context. Therefore, I had to skip this solver. The second one is Houdini’s native rigid body collision solver (Houdini RBD). Although this solver may often be powerful, as it can handle highly accurate collisions between geometry of any shape, it will considerably slow down the simulation time [Side FX 5].

In contrast, the Bullet Solver, a new default RBD engine in Houdini 12, is much faster than the Houdini RBD, can handle large data sets, and supports the “RBD Fractured Object” node [Side FX 5]. The only downside of this engine is the fact that it works only with convex meshes. For example, if one uses the “Cluster Pieces” parameter of the node to make the individual pieces into larger clusters, the Bullet makes overlapping collision boundaries between chunks due to newly created concave shapes (see figure 3.8).

![Overlapping Collision Boundaries](image)

Figure 3.8 – Overlapping collision boundaries between pieces

This may result in the explosive blowup of pieces at the beginning of the simulation. Instead, such cluster effects can be achieved by building Glue Constraints among rigid body objects within the dynamics level in Houdini. This topic will be discussed in more detail later in the paper.
4. Methodology

4.1 Object-Level Shattering

In Houdini, after cutting a mesh into bits using the “Voronoï Fracture” surface node, there is no way to select one piece among them in the viewport. The only thing that can be done is to create primitive groups for distinction (see figure 4.1).

![Figure 4.1 – Primitive groups](image)

In order for a user to select any of those pieces on the object level, the Python module inside my HDA converts each primitive group referring to a single chunk (created by the “Voronoï Fracture” node) into a separate object. Python scripts also create two sub-networks for the user, one of which is “INPUT” where any meshes that have been split reside and the other one is “OUTPUT” where split pieces that have been converted into separate objects reside (see figure 4.2).
Figure 4.2 – The object-level shattering workflow

Now the user will be able to select any chunk in the “OUTPUT” subnet and split it further as many times as they desire. Each new piece inside the “OUTPUT” subnet gains a unique name by adding a string such as “_1”, or “_2” to the name of its input mesh.

The original mesh whose name is “box” in the figure 4.2 is being split into three individual chunks whose names are “box_1”, “box_2”, and “box_3.” The output “box_3” is then used as the input when it gets split further. So its outputs become “box_3_1”, “box_3_2”, and “box_3_3.” Also, a custom attribute (“shatter_level”) is added to each piece and is updated during this fracturing process. Each time a mesh is split into smaller pieces, the attribute value of each new piece is incremented by one (see figure 4.3).

<table>
<thead>
<tr>
<th>outputs</th>
<th>shatter_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>box_1</td>
<td>1</td>
</tr>
<tr>
<td>box_2</td>
<td>1</td>
</tr>
<tr>
<td>box_3_1</td>
<td>2</td>
</tr>
</tbody>
</table>
Table 4.3 – A custom attribute “shatter_level”

This attribute plays a very important role as it affords the option to visualize fractured pieces according to the attribute’s value. The attribute is also used to assign different weights to those pieces in the dynamics context. I will further explain the details of dynamics weights in the following paragraph.

4.2 Automatic Dynamics Settings

The destruction tool allows all those split chunks to be easily imported into a dynamics simulation. Basically, the tool looks up a custom attribute named “dynamic_status” for each chunk. This attribute represents whether the piece is active (dynamic_status = 1) or passive (dynamic_status = 0), updated whenever a user clicks one of two buttons on the tool: “Make Active” or “Make Passive” (see figure 4.4).

Figure 4.4 – Make Active & Make Passive buttons

Those chunks whose “dynamic_status” values are zero will be included into a passive “RBD Fractured Object” dynamics node. On the other hand, those chunks whose
attribute values are one will be sent to different levels of active “RBD Fractured Object” nodes, according to the value of the “shatter_level” attribute of each chunk (see figure 4.5).

Figure 4.5 – A flowchart of how the tool brings fractured pieces into the dynamic level
The reason why I used multiple active “RBD Fractured Object” nodes is so the user can group similar-sized chunks together and assign different dynamic weights to each group. User parameters which allow one to set the weights (density) appear on the tool according to the number of active nodes (see figure 4.6).

![Figure 4.6 – Interactive dynamic-weights setting](image)

For example, if the user sets heavier weights for bigger pieces (lower “shatter_level” values), the result of the simulation can look more believable and interesting; larger chunks push smaller ones away rather than just stacking up on top of them.

### 4.3 Customizable Particle System

Countless debris bits along with smoke pour out when a building is collapsing in reality. However, in the field of visual effects, to put such numerous geometry bits into a rigid body simulation is neither efficient nor something computers can handle, despite the remarkable development of today’s computer technology. Instead, using a particle system to represent such debris can be a very effective way to add extra details onto the
destruction scene as simulation time of the particles is much faster than the RBD’s (see figure 4.7).

Figure 4.7 – An example of applying the particle system

A user can create such realistic debris effects using the particle system in my tool. There are three key steps for the particle creation. First, if the length of a velocity vector of any chunk exceeds a certain threshold value, some points that will be used to emit particles are appended to that chunk. At this moment, the tool uses the “inside” primitive group (newly created surfaces by the “Voronoi Fracture” node) in order for those points to be scattered on the interior parts of the chunk. This way, the user can make particles pour out from the inside of the building model (see figure 4.8).
Next, those scattered points start emitting particles. The tool uses all the chunks being collapsed as collision objects for the particles. The user can also adjust how much these particles will inherit the chunk’s velocity when released. Finally, the points that were generating particles disappear after a short while. This keeps particles from pouring out continuously. The destruction tool provides several parameters that allow the user to have control over all these features; the user can set the velocity threshold, the amount of emitters, and their emitting duration (see figure 4.9).

![Figure 4.8 – How the particle system works](image)

![Figure 4.9 – Parameters for the control of particles](image)
5. Conclusions / Future Work

5.1 Future Work

There are two significant options to be added to the destruction tool in the future. First, the tool has to have an option where a user can customize how to cluster pieces. Although it is possible for the user to have differently sized fractured pieces with my tool, he or she cannot help the regularity of shapes due to the “Voronoi Fracture” node. One possible solution is to use a “Glue Cluster” surface node (a built-in HDA) that allows one to set random glue constraints among pieces in the dynamics level within Houdini. Therefore, very irregular shapes can be achieved without the convex/concave issue of the Bullet Engine, discussed earlier. Secondly, it may be essential for users to extract transformation data of the simulated chunks with the tool and apply the data to their high-res models. An option that automates transferring such matrix data would not only save huge amounts of time but also render the details of the high-res model. This option would also let users avoid straight lines between fractured pieces due to the “Voronoi Fracture” node, as they can keep irregular edges on their high-res chunk models.

5.2 Conclusions

The object-level destruction tool, a Houdini Digital Asset supported by Python modules, gives visual effects artists a new way to deal with their destruction effects in a production environment. Allowing one to shatter based on selections in the object-level
makes the fracturing work much easier and more intuitive than working in the typical way: on the geometry level, which is time-consuming and difficult to edit. The tool conveniently automates all dynamics settings for rigid body simulations by simply turning any selected object into either an active or passive object. In addition to these, a customizable particle system can be added to create extra small debris or volumetric effects in a realistic fashion with my tool. It is my hope that with all these features, the new tool can help visual effects studios speed up their entire destruction pipelines without sacrificing realism or performance.
User Guidelines

When you click the “Shatter” button for the first time, the tool automatically creates the “DESTRUCTION” subnet. All other necessary nodes for your destruction work, along with “INPUT” and “OUTPUT” subnets, will be created within the “DESTRUCTION” node. You can see the shattered results or select them in the “OUTPUT” node.

Once you are satisfied with your fractured chunks, click the “Update Dynamics” button. This will create the “DYNAMICS” node and you will be able to see the simulation right away. Note that while two buttons, “Make Active” and “Make Passive”, can automatically update your simulation, the “Shatter” button cannot. So if you have created new chunks using the “Shatter” button, you need to click “Update Dynamics” again in order for those chunks to be seen in the simulation.

If you wish to add explosion effects to your destruction work, check the “Use Force” box on and specify your force object in the “Force Source” parameter. Note that your force object must be polygonal geometry as the tool uses its “point velocity” values.

Select the “DYNAMICS” node and click the “Add Debris” button to add particle effects. This will create the “DEBRIS” node. You can copy your own debris bits onto those particles or use them as the source for volumetric effects such as smoke or dust.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shatter</td>
<td>Shatter any selected object nodes. The split chunks can be seen in the “OUTPUT” subnet. You can also shatter multiple original meshes.</td>
</tr>
<tr>
<td>Hide/Unhide Selected</td>
<td>Hide selected object nodes from the viewport. It brings them back to the viewport when hit for a second time. This can be useful when you want to hide outer chunks to see inner ones or to shatter and split them further.</td>
</tr>
<tr>
<td>Number of Chunks</td>
<td>Determine how many pieces into which an object will be split. Note that a high value with many objects selected causes numerous chunk nodes to be created</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Display Method</td>
<td>Provide different options for displaying chunks. “Each Chunk” assigns random color to each chunk. “Shatter Level” picks color from the ramp, which pops up only when this option is selected, depending on the “shatter_level” attribute’s value of each chunk. “Active/Passive” assigns green color to active objects and dark gray to passive ones.</td>
</tr>
<tr>
<td>Hide Input/Output Nodes</td>
<td>Hide all nodes in INPUT and OUTPUT subnets if there are too many.</td>
</tr>
<tr>
<td>Add Interior Detail</td>
<td>Add extra edges with noise to interior primitives of each chunk. This directly uses the default “Add Interior Detail” option of the “Voronoi Fracture” node. Note that setting the “Noise Amplitude” parameter to more than zero may result in an explosion at the beginning of your simulation since the tool uses the Bullet solver. Refer to the convex/concave issue on 3.3 Bullet Solver.</td>
</tr>
<tr>
<td>Detail Size</td>
<td>Set the frequency of extra edges.</td>
</tr>
<tr>
<td>Noise Amplitude</td>
<td>Set the amount of noise.</td>
</tr>
<tr>
<td>Noise Frequency</td>
<td>Set the frequency of noise.</td>
</tr>
<tr>
<td>Use Dynamics</td>
<td>Allow you to have the RBD simulation.</td>
</tr>
<tr>
<td>Update Dynamics</td>
<td>Bring all fractured pieces in the “OUTPUT” subnet into the “DYNAMICS” dopnet and turn them into either active or passive.</td>
</tr>
<tr>
<td>Gravity</td>
<td>Set the magnitude of gravity for the RBD simulation.</td>
</tr>
<tr>
<td>Make Active</td>
<td>Once you’ve updated the dynamics after fracturing the original mesh, use this button to make the selected chunks in the “OUTPUT” node active. Make sure you set the “Display Method” parameter to “Active/Passive” in order to instantly see which one becomes active (green). Note that when you update dynamics for the first time, all fractured pieces become active.</td>
</tr>
<tr>
<td>Make Passive</td>
<td>Once you’ve updated the dynamics after fracturing the original mesh, use this button to make the selected chunks in the “OUTPUT” node passive. Make sure</td>
</tr>
</tbody>
</table>

so you will need to wait for a while.
you set the “Display Method” parameter to “Active/Passive” in order to instantly see which one becomes passive (dark gray).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Force</td>
<td>Allow you to add the explosive force to the RBD simulation.</td>
</tr>
<tr>
<td>Set Force</td>
<td>Hit this button only once as long as you have specified your force object in the “Force Source” parameter.</td>
</tr>
<tr>
<td>Force Source</td>
<td>Specify your force object.</td>
</tr>
<tr>
<td>Force Magnitude</td>
<td>Scale velocity values of points of your force geometry. Setting this value to negative usually produces a more violent result.</td>
</tr>
<tr>
<td>Create Particles</td>
<td>Allow you to have particles, used to make debris, dust, or smoke.</td>
</tr>
<tr>
<td>Add Debris</td>
<td>Create the “DEBRIS” node where particles can be created.</td>
</tr>
<tr>
<td>Number of Emitters</td>
<td>Determine how many emitting points to be scattered onto the interior parts of all chunks. Note that even after you set this parameter, the number of points will eventually decrease due to the “Emit Duration” parameter.</td>
</tr>
<tr>
<td>Minimum Velocity</td>
<td>Points will be scattered only when the velocity of each chunk exceeds this threshold value. Lowering this value will make particles created earlier.</td>
</tr>
<tr>
<td>Inherit Velocity</td>
<td>Determine how much particles inherit the velocity of their corresponding pieces.</td>
</tr>
<tr>
<td>Emit Duration</td>
<td>Determine how long each point will release particles from the first frame it created the initial particle.</td>
</tr>
<tr>
<td>Collide Tolerance</td>
<td>Distance from geometry at which particle is considered to have collided.</td>
</tr>
</tbody>
</table>
Bibliography


Tushevskiy, Vladislav. Email Interview. 11 Sept. 2012.