dust3d Documentation

Release 1.0.0-rc.1

Jeremyi HU

Feb 04, 2022
## Contents

1 Getting Started 1
   1.1 Download and Install Dust3D 1
   1.2 Dust3D Interface Overview 2
   1.3 Menu Bar 3
   1.4 Parts Tree Panel 7
   1.5 Script Panel 10
   1.6 Dust3D Shortcuts & Hotkey Guide 11
   1.7 Dust3D Script Reference 12

2 Dust3D Modeling Examples 23
   2.1 Modeling Ant using Dust3D 23
   2.2 Make a 3D model from scratch using Dust3D 26
   2.3 Modeling Camel using Dust3D 26
   2.4 Modeling Horse using Dust3D 27

3 For Developers 29
   3.1 Building Dust3D 29
   3.2 Write a 3D modeling software from scratch 32

4 Indices and tables 41
Getting Started

1.1 Download and Install Dust3D

- For Windows (64 Bit):
  
  https://github.com/huxingyi/dust3d/releases/download/1.0.0-rc.6/dust3d-1.0.0-rc.6.zip
  
  No need to install, unzip and run the exe.

- For Windows (32 Bit):
  
  https://github.com/huxingyi/dust3d/releases/download/1.0.0-rc.6/dust3d-1.0.0-rc.6-x86.zip
  
  No need to install, unzip and run the exe.

- For Mac OS X:
  
  https://github.com/huxingyi/dust3d/releases/download/1.0.0-rc.6/dust3d-1.0.0-rc.6.dmg
  
  If “The following disk images couldn’t be opened” popped up, that means the downloaded file was broken, please retry.

  If “can’t be opened because its integrity cannot be verified” popped up, please follow ni_kush’s answer in this reddit post.

  (Thanks @jimmygunawanapp for testing)

- For Linux:

  https://github.com/huxingyi/dust3d/releases/download/1.0.0-rc.6/dust3d-1.0.0-rc.6.AppImage

  $ chmod a+x ./dust3d-1.0.0-rc.6.AppImage
  $ ./dust3d-1.0.0-rc.6.AppImage

- Build on Your Own:

  Build from source.
1.2 Dust3D Interface Overview

1.2.1 1. Tool Box

The Tool Box contains tools that you use to switch the document edit mode, toggle the XYZ-axis and radius lock, indicate the mesh generating status, and trigger regenerate.

1.2.2 2. Document Title

You can have multiple windows, the document title shows what version of Dust3D you are using and the full path of the opened Dust3D document, if the document have unsaved changes, there will be a star(*) right after the path.

1.2.3 3. Dock Panel

Dock panel contains multiple tabs, including Parts Tree, Material List, Rig Weight View, Pose List, Motion List, and Script Panel.
1.2.4 4. Rendered Model

Rendered Model shows current generated mesh. Anytime, if there is a change would affect the mesh generation, the mesh would be generated automatically in realtime, and get rendered to Rendered Model after generated. You can use mouse Middle Button plus Shift key to rotate and move the Rendered Model. It’s floating above the canvas.

1.2.5 5. Node Front View

Every time you add a new node to canvas, there will be two items shows, the first one is the item you places, the second one is the automatically generated item for the same node but with different profile. If you add a separate node, which means it doesn’t connect with other nodes, the node is in front view, otherwise it depends on the view of the node you add from.

1.2.6 6. Node Side View

Side view is a profile heading left, see node front view for more details.

1.3 Menu Bar

1.3.1 File

New Window

Each window is a separate Dust3D Document. You can create a new window from here.

New

Delete all content in the current window and get ready for your new Dust3D Document in the same window.

Open…

Open a Dust3D Document, especially a file with extension(.ds3)

Save

Save current Dust3D Document.

Save As…

Same as Save except ask you provider a new save path.

Save All

Save all opened window.
Export...

Export Mesh only as Wavefront Format (.obj); Export Meshes, Skeleton, Textures and Animations as glTF Binary Format (.glb), you can use Don McCurdy’s online website, glTF Viewer https://gltf-viewer.donmccurdy.com/ to check the exported result; Export Meshes, Skeleton, Textures and Animations as Autodesk FBX Format(.fbx).

Change Reference Sheet...

Change the canvas background image, usually should be a turnaround reference sheet which has front and side view showed.

1.3.2 Edit

Add...

Switch to edit mode, get ready to add node.

Undo

Recover the last Document Snapshot from Undo Stack.

Redo

Reverse of Undo.

Delete

Delete selected nodes from canvas.

Break

Break selected edges, insert a new node in the middle of the edge.

Connect

Connect selected two nodes.

Cut

Copy then Delete selected nodes.

Copy

Copy selected nodes.
**Paste**

Paste nodes from Clipboard.

**H Flip**

Flip selected nodes horizontally.

**V Flip**

Flip selected nodes vertically.

**Rotate 90D CW**

Rotate selected nodes clockwise by 90 degrees. If you can hardly edit some nodes in normal front/side view, rotate it then edit it, and rotate it back after you finish editing.

**Rotate 90D CCW**

Rotate selected nodes counterclockwise by 90 degrees.

**Switch XZ**

Switch selected nodes’ X and Z position. If you accidentally put some parts in front view which you planned put into side view, you can select these nodes and switch the XZ components.

**Align To**

Align selected nodes with center anchor globally or selected nodes’ center locally. Normally, the center anchor (a Triangle) is not show up, you can turn on the Part Mirror to make it visible, then turn Part Mirror off, the center anchor would not gone once showed.

**Mark As**

Mark selected nodes as specified body location explicitly, such as Leg (Start), Leg (Joint), Leg (End), and Spine. This will help the rigging step generating more reasonable result.

**Select All**

Select all nodes. Each node have two profile items, only main profile get selected.

**Select Part**

Select all nodes which sit in the same part with the hovered or checked node.
Unselect All

Unselect all nodes.

1.3.3 View

Show Model

Usually, you will no need to use this, because the Rendered Model always show. But if you can not find the Rendered Model and you are sure the generation is done, then maybe it goes to some weird position, you can use this menu item to reset it’s position.

Show Parts List

The Parts List Panel is a tool window, if you closed it by accident, you can show it back here.

Toggle Wireframe

Rendered Model can be showed in two types, one with wireframe, one without.

Show Debug Dialog

This is for debug purpose only. It prints some useful information when debug.

1.3.4 Help

About

You can check the version info of Dust3D from here.

Fork me on GitHub

Dust3D is a totally free and open-sourced project, this bring you to the project website.

Report Issues

If you encounter any problem, or have any suggestion, thoughts, on Dust3D please drop it here, thanks.
1.4 Parts Tree Panel

In Dust3D, model consists of parts, part consists of nodes. User manipulates nodes’s position and radius, toggle part’s settings, then the mesh autogenerated by Dust3D.

1.4.1 Top Mini Buttons

(From left to right)

- Visible/(H)idden
  Toggle the part’s visibility on canvas, the mesh generation would not be affected. You can use it to hide some parts, then make edit other parts not been covered.

- (L)ock/Unlock
  Toggle the part’s edit lock. Locked part would still be visible on canvas, but can not been checked or edited.

- (J)oin/Not Join(to Final Mesh)
  Enable or disable the part, affects the final mesh generation. Unjoined part would not appear in the final result, only shows preview on Parts Tree Panel.

- (M)irror Modifier
  If there are two parts have the same shape but one sit left, the other sit right. It’s like the human body, you no need to make the arm part two times, just once, and mirror it.

- Color Picker
  Configure the part’s color and material settings. If there isn’t color chosen, the Rendered Model will be show as the configured preference color. The color chosen here will exported as a material color in the result output.

  Click the mini button, a tool widget will popup shows the chosen color, solubility, and material list.

  If one part configured color solubility, the generated color texture seam between the neighbor parts will be gradient filled using this part’s color with the specified solubility.

  Click the material preview image to choose material. Hold control while click to cancel the material selection.
1.4.2 Rendered Part Preview

Shows the rendered image of the whole part’s generated mesh. Current hovered or selected parts will be highlighted with a different gradient background color. If the mesh failed to generate, the rendered image will show as pure red color indicate you should adjust some nodes to avoid the generated mesh self intersection.

1.4.3 Bottom Mini Buttons

- Subdivide
  Toggle the subdivision of mesh generation. The generated mesh will be more smooth if this option is enabled.
- End Ro(U)ndable Modifier
  Toggle the begin/end roundable (Flat / Rounded). If you want the cut effect, choose not rounded, otherwise, the edge end will be rounded (Automatically added one more small face).
- Chamfer
  Enable this to bevel the edge.
- Cut Face
  Cut face is the extruding shape. Cut face could be configured as Quad, Pentagon, Hexagon, Triangle, and User Defined. You can turn a part to cut face from the part context menu by choosing target as Cut Face.
  The rotation parameter on the popup widget controls the rotation of the cut face along the edge direction.
- Thickness Modifier
  The are two parameters for this modifier, one controls the thickness along the base plane normal, the other controls the thickness along the perpendicular of base plane normal.
1.4.4 Context Menu

Right click on Parts List Panel to trigger Context Menu.

- **Base**
  Mesh layout and thickness adjustment are depend on base plane calculation. There are five type of options: Dynamic, Average, Side Plane, Front Plane, and Top Plane, see the demo below.
  - **Target**
    You can turn part as a cut face from here. Once a part becomes cut face, other parts can choose it as a extruding shape.
  - **Mode**
    Normal represents the normal mesh union, inverse represents the mesh subtraction. This release introduce Uncombined mode, which represents the component would not take part in the mesh boolean algorithms. Be noticed that, model with uncombined mode component exists, would not be generate as a watertight mesh.
    - **Hide / Show / Show All / Hide All**
      Check mini button **Visible/(H)idden**.
    - **Lock All / Unlock All**
      Check mini button **_ (L)ock/Unlock**.
      The process of mesh generation is mainly combining all the parts in listed order, so move up and down may affect the generated result. The order is especially important for inverse part been placed, if the inverse part sit in the first place of the list, the inverse operation would never happen, because nothing to invert before it.
      - **New Group / Move To**
Parts can be grouped as component, component can contain sub components. During the mesh generating progress, component’s mesh is been cached, so make multiple level of components can reduce the total mesh generating time.

### 1.5 Script Panel

Dust3D supports JavaScript language to generate the source document, which then been fed into the mesh generator to construct the final model. Please check [Dust3D Script Reference](#) for details.

#### 1.5.1 1. Rendered Model

Result model generated from the script.

#### 1.5.2 2. Script Editor

Paste your JavaScript code here, or type in the code directly. The script runner will be invoked automatically once there is any change detected from the script editor. If there is an error or console output from the script, the output will show under the editor.

#### 1.5.3 3. Variables Panel

You can create input control from script and receive the user input variable which can be used to adjust the settings for the procedural generation.
## 1.6 Dust3D Shortcuts & Hotkey Guide

### 1.6.1 Keyboard

<table>
<thead>
<tr>
<th>Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>- (Minus)</td>
<td>Decrease Radius</td>
</tr>
<tr>
<td>= (Equal)</td>
<td>Increase Radius</td>
</tr>
<tr>
<td>ALT + - (Minus)</td>
<td>Zoom Out Rendered Model</td>
</tr>
<tr>
<td>ALT + = (Equal)</td>
<td>Zoom In Rendered Model</td>
</tr>
<tr>
<td>, (Comma)</td>
<td>Rotate Multiple Nodes (Counterclockwise)</td>
</tr>
<tr>
<td>. (Period)</td>
<td>Rotate Multiple Nodes (Clockwise)</td>
</tr>
<tr>
<td>[ (Left Bracket)</td>
<td>Decrease Size</td>
</tr>
<tr>
<td>] (Right Bracket)</td>
<td>Increase Size</td>
</tr>
<tr>
<td>UP</td>
<td>Move UP</td>
</tr>
<tr>
<td>DOWN</td>
<td>Move Down</td>
</tr>
<tr>
<td>LEFT</td>
<td>Move Left</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Move Right</td>
</tr>
<tr>
<td>A</td>
<td>Toggle Select Mode and Add Mode</td>
</tr>
<tr>
<td>S</td>
<td>Enter Select Mode</td>
</tr>
<tr>
<td>D</td>
<td>Enter Paint Mode</td>
</tr>
<tr>
<td>G</td>
<td>Enter Mark Mode</td>
</tr>
<tr>
<td>CTRL + S</td>
<td>Save</td>
</tr>
<tr>
<td>CTRL + Z</td>
<td>Undo</td>
</tr>
<tr>
<td>CTRL + SHIFT + Y</td>
<td>Redo</td>
</tr>
<tr>
<td>CTRL + Y</td>
<td>Redo</td>
</tr>
<tr>
<td>CTRL + C</td>
<td>Copy</td>
</tr>
<tr>
<td>CTRL + V</td>
<td>Paste</td>
</tr>
<tr>
<td>CTRL + X</td>
<td>Cut</td>
</tr>
<tr>
<td>X</td>
<td>Toggle X-axis Edit Lock</td>
</tr>
<tr>
<td>Y</td>
<td>Toggle Y-axis Edit Lock</td>
</tr>
<tr>
<td>Z</td>
<td>Toggle Z-axis Edit Lock</td>
</tr>
<tr>
<td>H</td>
<td>Toggle Part Visible/(H)idden</td>
</tr>
<tr>
<td>J</td>
<td>Toggle Part (J)join/Not Join(to Final Mesh)</td>
</tr>
<tr>
<td>L</td>
<td>Toggle Part Edit Lock</td>
</tr>
<tr>
<td>M</td>
<td>Toggle Part X-axis-Mirror</td>
</tr>
<tr>
<td>B</td>
<td>Toggle Part Face Generating Type: (B)ox/Subdivisioned Box</td>
</tr>
<tr>
<td>U</td>
<td>Toggle Part End Roundable</td>
</tr>
<tr>
<td>C</td>
<td>Toggle Part Chamfer state</td>
</tr>
<tr>
<td>E</td>
<td>Switch the Selected Nodes to Different Profile (Main / Side)</td>
</tr>
<tr>
<td>F</td>
<td>Bring the part widget of current hovered part to visible area</td>
</tr>
<tr>
<td>W</td>
<td>Toggle render wireframe</td>
</tr>
<tr>
<td>O</td>
<td>Toggle render flat shading</td>
</tr>
<tr>
<td>R</td>
<td>Toggle canvas rotation</td>
</tr>
</tbody>
</table>
1.6.2 Mouse

<table>
<thead>
<tr>
<th>Button Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT BUTTON</td>
<td>Select / Move</td>
</tr>
<tr>
<td>CTRL + LEFT BUTTON</td>
<td>Multiple Select</td>
</tr>
<tr>
<td>CTRL + ALT + LEFT BUTTON</td>
<td>Multiple Unselect</td>
</tr>
<tr>
<td>SHIFT + LEFT BUTTON</td>
<td>Rotate / Pull (End Effector)</td>
</tr>
<tr>
<td>RIGHT BUTTON</td>
<td>Context Menu</td>
</tr>
<tr>
<td>MIDDLE BUTTON</td>
<td>Rotate Rendered Model</td>
</tr>
<tr>
<td>SHIFT + MIDDLE BUTTON</td>
<td>Move Rendered Model</td>
</tr>
<tr>
<td>ALT + LEFT BUTTON</td>
<td>Rotate Rendered Model</td>
</tr>
<tr>
<td>ALT + SHIFT + LEFT BUTTON</td>
<td>Move Rendered Model</td>
</tr>
<tr>
<td>WHEEL</td>
<td>Increase/Decrease Radius</td>
</tr>
<tr>
<td>CTRL + WHEEL</td>
<td>Increase/Decrease Size</td>
</tr>
<tr>
<td>SHIFT + WHEEL</td>
<td>Increase/Decrease Brush Size (on Paint Mode)</td>
</tr>
</tbody>
</table>

1.7 Dust3D Script Reference

1.7.1 Introduction

The most important function of Dust3D is to generate mesh from the source document, which contains the position, radius and connectivity information of all the nodes.

Normally, the nodes are drawn by user on the canvas. The scripting system, which we introduce in this document, is to provide a way to procedurally generate nodes, and then the mesh can be generated from these nodes.

Using nodes to generate mesh, already speed up the whole modeling progress. Scripting is to help speed up the speed up.

The following is the screenshot of the Script Panel, demonstrating how a tree been procedurally generated, you can find it from the Software Open Example Menu as Procedural Tree example.
1.7.2 Document Structure

The following two pictures shows the structure of Dust3D Document from different perspective.
1. Document
2. Canvas
3. Nodes and Edges
4. Part
5. Component

1.7.3 Document API

document.**createComponent**(parentComponent)

Create component and return the created component.

If parentComponent is omit, then component will be created as top level.

document.**createPart**(component)

Create part as child of component and return the created part.

The component is not optional.

document.**createNode**(part)

Create node as child of part and return the created node.

The part is not optional.

document.**connect**(firstNode, secondNode)

Connect two nodes by create a new edge between firstNode and secondNode.

firstNode and secondNode must come from the same part.

document.**setAttribute**(element, attributeName, attributeValue)

element can be component, part, node, and canvas.

document.**attribute**(element, attributeName)
Fetch the attribute value of element, name specified by attributeName

document.**createCheckInput**(controlName, defaultChecked)

Create a check input control, and return the value specified by user.

*defaultChecked can be true or false*

Return true when user checked the input box otherwise false.

document.**createFloatInput**(controlName, defaultValue, minValue, maxValue)

Create a float number input control, and return the value specified by user.

document.**createIntInput**(controlName, defaultValue, minValue, maxValue)

Create a int number input control, and return the value specified by user.

document.**createColorInput**(controlName, defaultColor)

Create a color picker, and return the color picked by user.

Returned value is a hex string represent the color, for example,

```
var leafColor = document.createColorInput("Leaf Color", 
   
   document.setAttribute(part, "color", leafColor);
```


document.**createSelectInput**(controlName, defaultSelectedIndex, optionsArray)

Create a select control, and return the selected index by user.

*optionsArray is a array list all the options, for example,*

```
var selectedModeIndex = document.createSelectInput("Mode", 0, modeList);
```

1.7.4 3D Math API

Dust3D introduces 3D vector math from three.js, the following API are supported: Vector3, Quaternion.

1.7.5 Document Element Attributes

Following are all the attributes grouped by element type, which can be used as the second parameter of *document.setAttribute* and *document.attribute*. Both attribute name and value are string format, even for boolean type, for example,

```
document.setAttribute(component, "expanded", "false")
```

**canvas**

There is only one canvas for a document, and you don’t need to create it, use *document.canvas* to fetch it.
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originX</td>
<td>The center anchor position.</td>
</tr>
<tr>
<td>originY</td>
<td></td>
</tr>
<tr>
<td>originZ</td>
<td></td>
</tr>
</tbody>
</table>

**rigType**

The only option currently supported is **Animal**

### component

You can have multiple depth of components. Use `document.createComponent` to create it.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>This name will be displayed on Parts Tree Widget</td>
</tr>
<tr>
<td>expanded</td>
<td>All components show as a tree widget. This attribute control if the tree item expand or not.</td>
</tr>
</tbody>
</table>
| combineMode | Options:  

  **Normal** This component’s generated mesh will combine with previous components’ generated mesh by union algorithm.  

  **Inversion** This component’s generated mesh will subtract the previous components’ generated mesh.  

  **Uncombined** This component’s generated mesh will merge with previous components’ generated mesh directly, without boolean algorithm. Use this option will cause the final generated mesh not watertight. |

### part

Part can’t be created directly from document, it must belong to a component. Use `document.createPart` to create it.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>Show nodes and edges on canvas if this attribute is true.</td>
</tr>
<tr>
<td>locked</td>
<td>Lock nodes and edges on canvas if this attribute is true.</td>
</tr>
<tr>
<td>subdived</td>
<td>The generated mesh will be subdivided if this attribute is true.</td>
</tr>
<tr>
<td>disabled</td>
<td>This part’s mesh will not be included in the final mesh result if this attribute is true, but still shows preview on Parts Tree Widget.</td>
</tr>
<tr>
<td>xMirrored</td>
<td>This part’s mesh will be mirrored on X axis if this attribute is true.</td>
</tr>
<tr>
<td>base</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td><strong>XYZ</strong> This value is actually showed as Dynamic on the UI.</td>
</tr>
<tr>
<td></td>
<td><strong>Average</strong></td>
</tr>
<tr>
<td></td>
<td><strong>YZ</strong> This value is actually showed as Side Plane on the UI.</td>
</tr>
<tr>
<td></td>
<td><strong>XY</strong> This value is actually showed as Front Plane on the UI.</td>
</tr>
<tr>
<td></td>
<td><strong>ZX</strong> This value is actually showed as Top Plane on the UI.</td>
</tr>
<tr>
<td></td>
<td>Please check the Script Panel for more introduction about Base.</td>
</tr>
<tr>
<td>rounded</td>
<td>An extra small sized face compared to the original one at the end effector will be generated if this attribute is true.</td>
</tr>
<tr>
<td>chamfered</td>
<td>The edge of the generated mesh will be beveled if this attribute is true.</td>
</tr>
<tr>
<td>target</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td><strong>Model</strong> The generated mesh of this part will be included in the final mesh result.</td>
</tr>
<tr>
<td></td>
<td><strong>CutFace</strong> The generated mesh of this part will be excluded in the final mesh result. The nodes coordinations of this part on the XY plane will be used as other parts’ Cut Face</td>
</tr>
<tr>
<td>cutRotation</td>
<td>Range: -1.0 ~ 1.0, represents the Rotation of Cut Face from -180 to 180 degrees.</td>
</tr>
<tr>
<td>cutFace</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td><strong>Quad</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Pentagon</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Hexagon</strong></td>
</tr>
</tbody>
</table>
**node**

Node can’t be created directly from document, it must belong to a part. Use `document.createNode` to create it.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>The node position on the canvas. The range of y is 0.0(Top) ~ 1.0(Bottom), range of x and z is 0.0(Left) ~ 1.0(Right)</td>
</tr>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
<tr>
<td>boneMark</td>
<td>Options: Used for rig generation.</td>
</tr>
<tr>
<td></td>
<td><strong>None</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Limb</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Tail</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Joint</strong></td>
</tr>
<tr>
<td>cutRotation</td>
<td>Range: -1.0 ~ 1.0, represents the Rotation of Cut Face from -180 to 180 degrees.</td>
</tr>
<tr>
<td>cutFace</td>
<td>Options:</td>
</tr>
<tr>
<td></td>
<td><strong>Quad</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Pentagon</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Hexagon</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Triangle</strong></td>
</tr>
<tr>
<td></td>
<td><strong>[partId]</strong> You can specified other part’s id as Cut Face. For example,</td>
</tr>
<tr>
<td></td>
<td>document.setAttribute(node, “cutFace”,</td>
</tr>
<tr>
<td></td>
<td>document.attribute(leafTemplatePart, “id”));</td>
</tr>
</tbody>
</table>

### 1.7.6 Procedural Tree Example

You can copy the following code to the script editor of Dust3D software to run it, scripting function requires 1.0.0-beta.22 and above.
```
dstaking Documentation, Release 1.0.0-rc.1

var SUBDIVED = document.createCheckInput("Subdived", false);
var MAX_RADIUS_OFFSET = document.createFloatInput("Max Radius Offset", 0.10, -1.0, 1.0);
var MAX_ORIGIN_OFFSET = document.createFloatInput("Max Origin Offset", 0.47, -1.0, 1.0);
var TREE_HEIGHT = document.createFloatInput("Height", 0.85, 0.1, 1.0);
var TRUNK_RADIUS = document.createFloatInput("Trunk Radius", 0.02, 0.01, 0.2);
var TRUNK_SEGMENTS = document.createIntInput("Trunk Segments", 15, 2, 30);
var TRUNK_COLOR = document.createColorInput("Trunk Color", "#939297");
var BRANCH_MIN_ANGLE = document.createFloatInput("Branch Min Angle", 10.72, 8, 30);
var BRANCH_MAX_ANGLE = document.createFloatInput("Branch Max Angle", 58.81, 50, 75);
var MAX_BRANCHES_PER_GROWTH = document.createIntInput("Max Branches", 7, 1, 7);
var BRANCH_LENGTH_MIN_RATIO = document.createFloatInput("Branch Length Min Ratio", 0.24, 0.2, 0.9);
var BRANCH_LENGTH_MAX_RATIO = document.createFloatInput("Branch Length Max Ratio", 0.78, 0.2, 0.9);
var LEAF_MIN_LENGTH = document.createFloatInput("Leaf Min Length", 0.04, 0.005, 0.1);
var LEAF_MAX_LENGTH = document.createFloatInput("Leaf Max Length", 0.08, 0.005, 0.2);
var LEAF_SEGMENTS = document.createIntInput("Leaf Segments", 5, 3, 6);
var LEAF_COLOR = document.createColorInput("Leaf Color", "#58834b");

var X = new THREE.Vector3(1, 0, 0);
var Y = new THREE.Vector3(0, 1, 0);

function createNode(part, origin, radius)
{
    var node = document.createNode(part);
    document.setAttribute(node, "x", origin.getComponent(0));
    document.setAttribute(node, "y", origin.getComponent(1));
    document.setAttribute(node, "z", origin.getComponent(2));
    document.setAttribute(node, "radius", radius);
    return node;
}

THREE.Vector3.prototype.random = function (maxOffset)
{
    for (var i = 0; i < 3; ++i)
        this.getComponent(i) + maxOffset * Math.random();
    return this;
};

function randInRange(min, max)
{
    if (min > max) {
        var tmp = min;
        min = max;
        max = tmp;
    }
    return (min + (max - min) * Math.random());
}

function createLeafCutFace()
{
    var component = document.createComponent();
    var part = document.createPart(component);
    document.setAttribute(part, "target", "CutFace");
    var predefinesCutFace = [
```
{radius: "0.005", x: "0.314637", y: "0.336525", z: "0.713406"},
{radius: "0.005", x: "0.343365", y: "0.340629", z: "0.713406"},
{radius: "0.005", x: "0.377565", y: "0.340629", z: "0.713406"},
{radius: "0.0159439", x: "0.402189", y: "0.341997", z: "0.713406"},
{radius: "0.005", x: "0.440492", y: "0.329685", z: "0.713406"},
{radius: "0.005", x: "0.473324", y: "0.317373", z: "0.713406"},
];

var previousNode = undefined;
for (var i = 0; i < predefinesCutFace.length; ++i) {
  var node = createNode(part,
    new THREE.Vector3(parseFloat(predefinesCutFace[i]["x"]),
    parseFloat(predefinesCutFace[i]["y"]),
    parseFloat(predefinesCutFace[i]["z"])),
    parseFloat(predefinesCutFace[i]["radius"]));
  if (undefined != previousNode)
    document.connect(previousNode, node);
  previousNode = node;
}
return part;
}

function getRandChildDirection(parentDirection)
{
  var rotationAxis = parentDirection.clone().cross(X);
  var degree = randInRange(BRANCH_MIN_ANGLE, BRANCH_MAX_ANGLE);
  var branchRotateQuaternion = new THREE.Quaternion();
  branchRotateQuaternion.setFromAxisAngle(rotationAxis, degree * (Math.PI / 180));
  var rotatedDirection = parentDirection.clone().applyQuaternion(branchRotateQuaternion);
  var distributeQuaternion = new THREE.Quaternion();
  distributeQuaternion.setFromAxisAngle(Y, 360 * Math.random() * (Math.PI / 180));
  rotatedDirection = rotatedDirection.applyQuaternion(distributeQuaternion);
  return rotatedDirection;
}

var CUTFACE_PART_ID = document.attribute(createLeafCutFace(), "id");

function createLeaf(leafRootPosition, parentDirection)
{
  var direction = getRandChildDirection(parentDirection);
  var component = document.createComponent();
  var part = document.createPart(component);
  document.setAttribute(part, "color", LEAF_COLOR);
  document.setAttribute(part, "cutFace", CUTFACE_PART_ID);
  var length = randInRange(LEAF_MIN_LENGTH, LEAF_MAX_LENGTH);
  var segments = LEAF_SEGMENTS;
  var maxRadius = length / 3;
  var toPosition = leafRootPosition.clone().add(direction.clone().multiplyScalar(length));
  var previousNode = undefined;
  for (var i = 0; i < segments; ++i) {
    var alpha = (i + 0.0) / segments;
    var origin = leafRootPosition.clone().lerp(toPosition, alpha);
    var radiusFactor = 1.0 - 2 * Math.abs(alpha - 0.3);
    var radius = maxRadius * radiusFactor;
var node = createNode(part, origin, radius);
if (undefined != previousNode)
    document.connect(previousNode, node);
previousNode = node;
}

test function createBranch(branchRootPosition, radius, length, segments, parentDirection, maxRadius)
{
    var rotatedDirection = getRandChildDirection(parentDirection);
    var branchEndPosition = branchRootPosition.clone().add(rotatedDirection.multiplyScalar(length));
    createTrunk(branchRootPosition, radius, branchEndPosition, segments, maxRadius);
}

function shouldGrowBranch(alpha, alreadyGrowedNum)
{
    var factor = alpha * Math.random();
    if (factor > 0.5 && factor < 0.8) {
        if (alreadyGrowedNum > 0)
            return true;
        if (Math.random() > 0.5)
            return true;
    }
    return false;
}

function shouldGrowLeaf(alreadyGrowedNum)
{
    var factor = Math.random();
    if (factor > 0.5) {
        if (alreadyGrowedNum > 0)
            return true;
        if (Math.random() > 0.5)
            return true;
    }
    return false;
}

function createTrunk(fromPosition, fromRadius, toPosition, segments, maxRadius)
{
    var component = document.createComponent();
    var part = document.createPart(component);
    if (SUBDIVED)
        document.setAttribute(part, "subdived", "true");
    document.setAttribute(part, "color", TRUNK_COLOR);
    var previousNode = undefined;
    var toRadius = fromRadius * 0.2;
    var direction = toPosition.clone().sub(fromPosition).normalize();
    var length = fromPosition.distanceTo(toPosition);
    for (var i = 0; i < segments; ++i) {
        var alpha = (i + 0.0) / segments;
        var origin = fromPosition.clone().lerp(toPosition, alpha);
        var radius = fromRadius * (1.0 - alpha) + toRadius * alpha;
        radius += radius * MAX_RADIUS_OFFSET * Math.random();
        if (undefined != maxRadius && radius > maxRadius)
radius = maxRadius;
var oldY = origin.y;
origin.random(length * 0.1 * MAX_ORIGIN_OFFSET);
origin.setY(oldY + radius * 0.5 * Math.random());
if (undefined != maxRadius && i == 0)
  origin = fromPosition;
var node = createNode(part, origin, radius);
if (undefined != previousNode)
  document.connect(previousNode, node);

var maxBranches = MAX_BRANCHES_PER_GROWTH * Math.random();
var alreadyGrewedBranchNum = 0;
var alreadyGrewedLeafNum = 0;
for (var j = 0; j < maxBranches; ++j) {
  if (shouldGrowBranch(alpha, alreadyGrewedBranchNum)) {
    var branchLength = length * randInRange(BRANCH_LENGTH_MIN_RATIO,
                                            BRANCH_LENGTH_MAX_RATIO);
    var branchRadius = radius * 0.5;
    var branchSegments = segments * (branchLength / length);
    if (branchSegments >= 3 && branchRadius > TRUNK_RADIUS / 20) {
      createBranch(origin, branchRadius, branchLength, branchSegments,
                    direction, radius);
      ++alreadyGrewedBranchNum;
    }
  }
  if (undefined != maxRadius && shouldGrowLeaf(alreadyGrewedLeafNum)) {
    createLeaf(origin, direction);
    ++alreadyGrewedLeafNum;
  }
}
previousNode = node;

var treeTopPosition = new THREE.Vector3(0.5, 1.0, 1.0);
var treeTopPosition = treeRootPosition.clone();
treeTopPosition.add((new THREE.Vector3(0, -1, 0)).multiplyScalar(TREE_HEIGHT));
document.setAttribute(document.canvas, "originX", 0.506767);
document.setAttribute(document.canvas, "originY", 0.615943);
document.setAttribute(document.canvas, "originZ", 1.08543);
createTrunk(treeRootPosition, TRUNK_RADIUS, treeTopPosition, TRUNK_SEGMENTS);
2.1 Modeling Ant using Dust3D

2.1.1 Prepare Reference Sheet

You can start modeling without any reference, however, if you model on top of a turnaround reference sheet, the final result would be better, it also ease the progress of modeling. The recommended features for reference sheet in Dust3D:

- Only one image, but include at least two profiles in the image, one for front view, one for side view. And align horizontally with the same height.
- Image size less than 1MB. Because if you save the document, the reference image will also be included in the .ds3 file.
- No blur filter, make a clean sheet, Dust3D will blur it for you when you load the image.
2.1.2 Modeling in Dust3D

1. Load Reference Sheet from Menu: File > Change Turnaround...
2. Click Plus(+) Icon from Tool Box
3. Place node on the canvas according to the reference background.
4. Use Mouse Wheel to Adjust the radius of the node, hold shift to slow down the scroll scale.
5. Use the Parts List panel to toggle the settings to make you easily select or see nodes.
6. Use Mouse Middle button plus Shift key to rotate and move around the Rendered Model.
2.1.3 Export Mesh

After finished modeling, Export the result mesh from Menu: File > Export...

2.1.4 Download

You can download all the resources including the full Dust3D Document from the following links.


2.1.5 Turnaround Reference Resource

1. https://www.antweb.org/
2. https://drawingdatabase.com/

(If you know more resource, please edit this page in Github to share, thanks.)
2.2 Make a 3D model from scratch using Dust3D

2.2.1 Download

You can download all the resources including the full Dust3D Document from the following links.


2.3 Modeling Camel using Dust3D

2.3.1 Download

You can download all the resources including the full Dust3D Document from the following links.

2.4 Modeling Horse using Dust3D

2.4.1 Download

You can download all the resources including the full Dust3D Document from the following links.

3.1 Building Dust3D

3.1.1 Overview

The UI of Dust3D is built on Qt5, the third party dependencies which may need to be compiled separately are the CGAL and Instant Meshes, however, CGAL will introduce some new dependencies, such as boost and gmp library.

Prerequisites

- Qt
  https://www.qt.io
- CGAL
  https://www.cgal.org
- Instant Meshes
  https://github.com/wjakob/instant-meshes
- Boost
  https://www.boost.org
- CMake
  https://cmake.org/

Building

Here are the snapshots of the build commands on different platforms, you may use customized defines on your system. If you encounter build issues, please follow the ci files step by step,
Windows

The following steps are for windows 64 bit, please refer to the appveyor.yml file for windows 32 bit

Install Qt 5.13.2

Install Visual Studio 2017 Community edition

Install CMake, during installation, choose Add CMake to the system PATH for the current user

Download boost 1.66 and extract to C:\Libraries make sure you got your boost path on C:\Libraries\boost_1_66_0

Download Dust3D Source Repository and extract to your desktop as C:\Users\IEUser\Desktop\dust3d, please note the IEUser should be replaced with your windows user name.

Build CGAL as the following steps:

```
From Start Menu, Open x64 Native Tools Command Prompt for VS 2017:
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community>cd
  -C/Users/IEUser/\Desktop\dust3d
C:\Users/IEUser/\Desktop/\dust3d>cd thirdparty/\cgal\CGAL-4.13
C:\Users/IEUser/\Desktop/\dust3d/thirdparty/\cgal\CGAL-4.13>mkdir build
C:\Users/IEUser/\Desktop/\dust3d/thirdparty/\cgal\CGAL-4.13>cd build
C:\Users/IEUser/\Desktop/\dust3d/thirdparty/\cgal\CGAL-4.13\build>cmake -G "Visual
  Studio 15 2017" -A x64 -DCMAKE_BUILDTYPE=RelWithDebInfo -DCGAL_DIR=C:\Users\IEUser\Desktop\dust3d/thirdparty/\cgal\CGAL-4.13/boost
  C:\Users/IEUser/\Desktop/\dust3d/thirdparty/\instant-meshes/\build>cmake -G "Visual
  Studio 15 2017" -A x64 -DLIBRARY="C:\Libraries\boost_1_66_0"
C:\Users/IEUser/\Desktop/\dust3d/thirdparty/\instant-meshes/\build>msbuild /p:Configuration=Release ALL_BUILD.vcxproj
```

Build Instant Meshes as the following steps:

```
```

Build Dust3D as the following steps:

```
```

After dust3d.exe is been built in C:\Users\IEUser\Desktop\dust3d\release, copy the following files to release folder:

```
C:\Users/IEUser/\Desktop/\dust3d/thirdparty/\instant-meshes/\build/\Release/\instant-meshes.dll
```
### Mac

**Outdated, help needed**

```shell
$ cd /Users/jeremy/Repositories/dust3d
$ qmake -spec macx-xcode
Open dust3d.xcodeproj in Xcode and build
```

### Ubuntu

**Outdated, help needed**

```shell
;Install Qt5
$ sudo apt-get install --reinstall qtchooser
$ sudo apt-get install qtbase5-dev

;Prepare compile environment for CGAL-4.13
$ sudo apt-get install libcgal-dev ; This is not the latest version, will encounter --compiler error when build the Dust3D with this version, but helps resolve internal --dependencies of CGAL for you
$ sudo apt install cmake

;Install CGAL-4.13
$ unzip CGAL-4.13.zip
$ cd CGAL-4.13
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install

;Clone the Main project
$ cd ~/Documents
$ git clone https://github.com/huxingyi/dust3d.git

;Compile Dust3D
$ cd ~/Documents/dust3d
```

### 3.1. Building Dust3D

(continues on next page)
3.2 Write a 3D modeling software from scratch

The origin and the future of Dust3D

3.2.1 Origin

I want to write a 3D modeling software since 2015, back to that time, I was trying to make a 2.5D MMORPG game. I self-learned blender for a while from youtube, the blender is quite good actually, however, the workload made me realized that it is impossible for one person to make tons of models, texture them, rigidify them, animate them, then use them in game, because built a simple dinosaur model took me half day.

I watched lots of tutorials about how to quickly make a game ready model, tried to figure out a unified way, a repeatable mode, which can be simplified in programming language. I summarized most usual steps of making a model: firstly, setup a turnaround reference sheet for front view, side view, and back view, secondly, make a plane then subdivide to hexagon, extrude this hexagon by following the reference sheet, adjust the size of the face, fine-tune in different angle, back and forth, finally get the base model.

3.2.2 Initial Experiment

Looks like I can write a software to automatically do these steps for me, I fed it reference sheet, it comes out a model. Let’s do it, I made a very rough test program, to recognize each view in the image, extract the boundaries, extrude faces according to the boundaries, it works if you zoom out the final result to a teeny tiny size. But it’s too small to be used in a game.

3.2.3 New Idea

Some day, I googled some monster generation keywords, and found Jimmy Gunawan’s blog, I was shocked by his article, this is what I am looking for, this is the answer, I was very excited and when dug into the technonigies behind blender’s Skin Modifier which described in Jimmy’s blog, I found the paper: <B-Mesh: A Fast Modeling System for Base Meshes of 3D Articulated Shapes>, at that point, I thought it’s time to finalize the idea to a real working software.
3.2.4 Launch

I launched the Dust3D project, and post my plan on reddit even I haven’t done much things. I did this because, as a newbie in game industry, I don’t want miss somethings in the very beginning. Thanks the amazing Redditors, I learned lots of new software names and modeling terms, such as Meshmixer, CGAL, and so on.

3.2.5 Reinventing the wheel

Reinventing the wheel is fun, so I didn’t strictly follow the advise by using the existing libraries, I want make a 3D software from scratch. I want draw the 3D world vertex by vertex, line by line. This is the first screenshot of Dust3D, using raw OpenGL without any dependency except OpenGL environment:

![First screenshot of Dust3D](image.jpg)

Very quickly, I found it took so much time to debug the drawing issues, so I carefully introduced GLU library, this is what it looks like:
After some time, I thought Qt is much easy to use, so I carefully introduced Qt. And then the Bmesh algorithm implemented:
Catmull-Clark subdivision:
Now, it’s time to make a more formal UI.

3.2.6 Reinventing the wheel Again

You can see, the repository start from zero dependency, and then introduced some things inevitably. This way goes well if continues. But there is something happened. Because of no complex UI, I use blender to build the relationships of Bmesh balls, and I found a bug of blender in the Callada exporter, I tried to fix it by myself, so I downloaded blender source, fix it then submitted a patch. In this progress, I was sick of different C++ version problems, So I decided to remove all the C++ code from my Dust3D codebase.

Qt is C++, so Qt removed. I tried to find some UI library instead, Dear ImGui was promising, but because it’s C++, so abandoned. I started the UI from zero again, this is what it looks like:
Aha!

### 3.2.7 Gap

I started Dust3D project in Australia when I was on a work and holiday visa. There are lots of things stopped me from coding on this project, it’s quite busy. This made me to rethink the decisions I made. Remove all the dependencies are not good, I am making a 3D modeling software, not a GUI library. I also started to think about some details of the modeling progress. In the Bmesh paper, the author expressed some limitations that it is not suitable for making sharp edges. We all known, when we make a model for game, inevitably, there will be a cloth something, definitely makes some sharp shapes.

### 3.2.8 Reinvestigate

I checked almost all the modeling software through youtube tutorials videos, tried to find how they do the job. these software including Houdini, I was shocked by the node based modeling technologies. I thought this is what I want, this is the answer, Looks familiar, right? :-)

### 3.2.9 Restart

I created a new branch named poc to do the proof of concept. Not by exactly implementing the node based modeling, I tried to define a new modeling script language, it can be easily embed in command line. At that time, I built many fundamental mesh operating algorithms, such as chamfer a mesh, mesh booleans.
3.2.10 Rust

I can’t remember the exact reason, maybe the project name? anyway, I was distracted by the rust language. I tried rewrite all the fundamental mesh algorithms in Rust to practice the language skills. This is how the meshlite library came out.

3.2.11 Finalize

Now, I had far much better understanding of mesh, and know how to generate the mesh I want, no matter it is smooth or sharp. After finish the meshlite library, I tried to build the UI again. There are not so much choice of UI framework in the Rust world. And I did some investigation, and played many GUI solutions, such as bgfx, I even fixed a trivial issue of bgfx and got it merged. But finally, I still decided to use Qt. This time, the whole coding progress is very smooth, Qt for UI, Rust for algorithm, worked like a charm, and Rust never crash on right use case, what I mean by saying that, rust have some built-in difficulty to build a double-linked like data structure, so I need some unsafe code or index based system to support the multiple linked data, such as the famous half-edge structure in mesh processing, because the index based system are not protected by the Rust language, sometimes, it crashes on some logic error. I found I am happy with Rust and C++11 and the new Qt signal slots, I also happily introduced Carve and CGAL libraries to do the mesh union.

3.2.12 Current

Today, I decide to share my story, I have finished the stage one of Dust3D. It’s not perfect, but it is what I thought it should like in many years ago. This is what I want, this is the answer for my past years.
3.2.13 Future

Currently, there is no auto unwrap texture, no auto rigidify, no auto animation generated. There is a far way to go, and I am looking forward to it. Thanks for reading.

PS: Auto-unwrap texture and auto-rigidify already landed, auto-animation is under development. (May 30, 2018)
CHAPTER 4

Indices and tables

- genindex
- modindex
- search