BuildSurface in Besiege

Chapter 1 Vertex and Normal

1.1 Buildsurface's control points
1.1.1 Read file
1.1.2 Save points
1.2 Control points transformation
1.3 Generate Mesh

Chapter 2 Material

2.1 Set material color
2.2 Generate UVmap

Chapter 1 Vertex and Normal

1.1 Buildsurface's control points

1.1.1 Read file

In .bsg files, a BuildSurface is composed of three kinds of blocks: BuildNodes, BuildEdges and BuildSurfaces. They can be searched by their guid, and connect to each other.

Let's see BuildNodes first.

```
<Block id="71" guid="96169a57-b99f-400a-b35f-a6d0b151c063">
    <Transform>
        <Position x="0" y="0.5" z="0" />
        <Rotation x="-0.7071068" y="0" z="0" w="0.7071068" />
        <Scale x="1" y="1" z="1" />
    </Transform>
    <Data />
</Block>
```

It's block id is 71, have a guid and a position. Other values are useless here.

Each block's guid is an unique code in a single .bsg file. It seems like some kind of Hash, but I'm not sure how it works. So I just save guids as strings.

Here is the structure of BuildNodes.

Block Name(string)	BuildNode
Block id(int)	71
guid(string)	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Position(VECTOR)	x,y,z

Then comes to BuildEdge.

```
<Block id="72" guid="20df0bd2-f09f-420a-917f-26796fb47396">
    <Transform>
        <Position x="0" y="0.5000002" z="-1" />
        <Rotation x="0" y="1" z="1.192092E-07" w="0" />
        <Scale x="1" y="1" z="1" />
    </Transform>
    <Data>
        <String key="start">96169a57-b99f-400a-b35f-a6d0b151c063</String>
        <String key="end">f0b32dc7-604b-42a2-84b0-8edc071759da</String>
    </Data>
</Block>
```

BuildEdge is similar to BuildNode. The difference is BuildEdge got two more guid, "start" and "end". These two guids saved edge's start node's guid and end node's guid.

Here is the structure of BuildEdges.

Block Name(string)	BuildEdge
Block id(int)	72
guid(string)	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
start(string)	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
end(string)	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Position(VECTOR)	x,y,z

Finally, the BuildSurface.

```
<Block id="73" guid="2e367e82-7f45-4813-a80e-a68f7ebb967b">
    <Transform>
        <Position x="0" y="0.5" z="0" />
        <Rotation x="-0.7071068" y="0" z="0" w="0.7071068" />
        <Scale x="1" y="1" z="1" />
    </Transform>
    <Data>
        <Integer key="bmt-surfMat">0</Integer>
        <Boolean key="bmt-painted">False</Boolean>
        <Boolean key="bmt-aero">True</Boolean>
        <Color key="bmt-hue">
            <R>0.5</R>
            <G>0.2</G>
            <B>0.2</B>
        </Color>
        <Single key="bmt-sat">0.6</Single>
        <Single key="bmt-lum">0.5</Single>
        <Single key="bmt-thickness">0.08</Single>
        <String key="edges">20df0bd2-f09f-420a-917f-26796fb47396|27796a31-1f96-4c91-8665
        <Integer key="materialIndex">1</Integer>
    </Data>
</Block>
```

BuildSurface's transform values are totally useless(Unless it was transformed by HardScale.), just throw them away.

"bmt-surfMat" controls BuildSurface's material, 0 is wood, 2 is glass. Seems there are more than 2 materials, maybe metal?

"bmt-painted" shows whether the BuildSurface was painted. False means it wasn't, and True means it was.

"bmt-aero" is about aerodynamic calculation, play none business here.

"bmt-hue" defined the color of BuildSurface, by using sRGB, only works while "bmt-painted" is True.

"bmt-sat" is saturation, and "bmt-lum" is luminosity(or lightness). they also only work while "bmt-painted" is True.

"bmt-thickness" is the thickness, it should not larger than 0.2 or smaller than 0.005, or the game will replace it by 0.2 or 0.005.

"edges" are the edges' guids, separated by '|'. it'll contain 3 or 4 guids. If it only contain 3 guids, the BuildSurface is not a quad.

"materialIndex" is useless. It just shows whether the BuildSurface is a quad or not, which you already know during searching edges.

Here is the structure of BuildSurfaces.

Block Name(string)	BuildSurface
bmt-surfMat(int)	0|2
bmt-painted(bool)	True|False
bmt-hue(VECTOR)	R[0.0,1.0],G[0.0,1.0],B[0.0,1.0]
bmt-sat(float)	[0.0,1.0]
bmt-lum(float)	[0.0,1.0]
bmt-thickness(float)	[0.005,0.2]
edges(string)	guid|guid|guid(|guid)

1.1.2 Save points

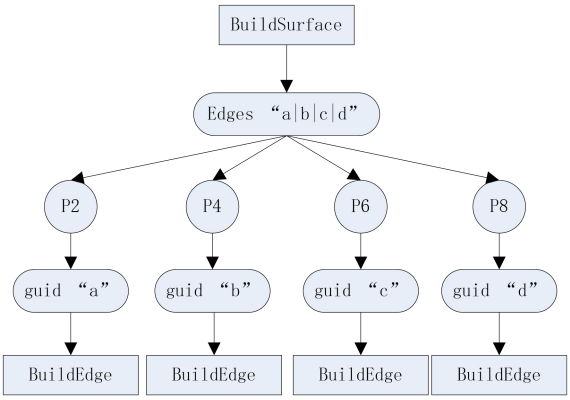
As we know, a BuildSurface has 8 control points, 4 nodes and 4 edges. To easier explain how to search these control points' position, define these control points as table 1-1 first.

P1	P8	P7
P2	NULL	P6
P3	P4	P5

Table 1-1

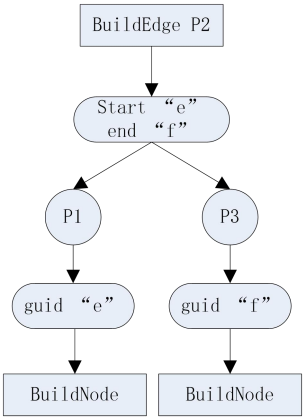
P1, P3, P5, P7 are nodes, there position can be find in BuildNodes, and P2, P4, P6, P8 are edges, there position can be find in BuildEdges. Since BuildSurface have no midpoint, the center of table 1-1 is NULL. If a BuildSurface have only 3 edges, which means it doesn't have P7 and P8's guids, we can simply treat P1, P8, P7 as same point

Edges' positions could be easily found by searching BuildSurface's edges. For example, a BuildSurface's edges is "a|b|c|d", it means P2's guid is "a", P4's guid is "b", P6's guid is "c", and P8's guid is "d". Then you search BuildEdges that have the same guid P2, P4, P6, P8 have, and get their positions. Just like picture 1-1.



Picture 1-1

Nodes' positions could be find by searching BuildEdge's start and end. For example, P2's start is "e" and end is "f", it means P1's guid is "e", and P3's guid is "f". Then you search BuildNodes that have the same guid P1, P3 have. Just like picture 1-2.



Picture 1-2

Watch out, a BuildEdge could be used by multi BuildSurface, and a BuildNode could also be used by multi BuildEdge. So in most of time, a BuildSurface's edges may not sequence saved. If the program just search edges and nodes sequencialy, BuildEdges' may not appropriately connected with each other.

1.2 Control points transformation

Now we have all control points. But BuildSurface is a Bezier surface, which means for each edge, there should not have control points on the edge. To easier explain how to transport these control points' position, define a new control points map as table 1-2 first.

B1	B8	B7
B2	B9	B6
B3	B4	B5

Table 1-2

Use P1~P8 in table 1-1 to generate new contral points.

Keep nodes as they are.

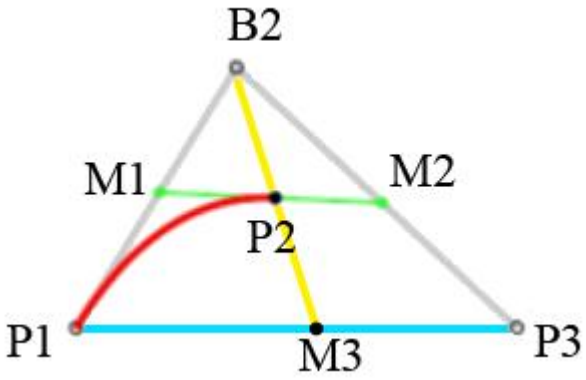
B1=P1

B3=P3

B5=P5

B7=P7

Then comes to edge. Use P1-P2-P3 as the example. Considering about the behavior of BuildSurfaces' resolution in Besiege, P2 is the midpoint of curve P1-P2-P3. Based on the characteristic of Bezier curves, we get picture 1-3.



Picture 1-3

∵P2 is the midpoint of the curve P1-P2-P3

∴P2 is the midpoint of M1M2(characteristic of Bezier curve)

∴M1 is the midpoint of P1B2, M2 is the midpoint of B2P3(characteristic of Bezier curve)

∴▲B2M1M2~▲B2P1P3, |B2M1|=|M1P1|

Set M3 as the midpoint of P1P3

∵▲B2M1M2~▲B2P1P3, P2 is the midpoint of M1M2, M3 is the midpoint of P1P3

∴B2, P2, M3 on the same straight line

∴▲B2M1P2~▲B2P1M3

∵|B2M1|=|M1P1|

∴|B2P2|=|P2M3|

∴B2=P2+(P2-M3)=2*P2-M3=2*P2-(P1+P3)/2

Do the same thing with B4, B6, B8.

B4=2*P4-(P3+P5)/2

B6=2*P6-(P5+P7)/2

B8=2*P8-(P7+P1)/2

Now considering B9. A Bezier surface at least have 9 control points, while a BuildSurface only have 8. The last control point is calculated by using B1~B8.

The empirical formula is here.

$$B9=2*(B2+B4+B6+B8)/4-(B1+B3+B5+B7)/4$$

Just a variant of the formula used in Bezier curve. Lazy Spiderling(

1.3 Generate Mesh

Now, you have all the control points of a Bezier surface. Use them to generate a Bezier surface, and Extrude Faces Along Normals both side, offset=BuildSurface.thickness/2. Then you get what you want.

Chapter 2 Material

2.1 Set material color

A BuildSurface has 5 values about the color, RGB, sat and lum.

RGB works in the same way as other blocks. Though it seems like HSV in game, it is sRGB in fact.

Sat is saturation, nothing more to say.

About lum, I'm not sure whether it controls ambient or diffuse.

If surfMat=2, the material should be glass, don't forget set the alpha. You can find glass textures by using AssetStudio to unpack the game.

2.2 Generate UVmap

Not sure about how Spiderling generate BuildSurface's UVmap, so I used my own way.

Examples here.