情報可視化論
最終課題解説資料

提出日：H28/6/13 (Mon.)

学籍番号：169X025X

氏名：吉川 友真

1　作成したファイルについて

私が作成したファイルの全体図とユーザーインターフェース (UI) 部分の拡大図をそれぞれ図 1, 2 に示す．UI では radio button を多く用いて作成している．また，注意事項や項目の意味などを UI 上で簡単に説明しておくことで，誰にでも扱いやすくなるよう気をつけた．適用するには，Apply ボタンを押せば更新される．何が変更できるかは，UI に示す各項目の補足情報 (英文) を見ることで理解できると考えるが，それぞれの動作内容については，2 章の追加した機能の項目で簡単に解説する．また，説明に必要なソースファイルを Listing 1 〜 3 に示すが，実際に動作させるには，授業中に指定された他の多くのファイルも必要となる．

2　追加した機能

追加した機能は，「色の付け方の選択」，「色指定値の入力フォーム」，「シェーディングの種類選択」，「リフレクションモデルの種類選択」，「周囲の立方体の枠の表示非表示選択」の 5 つである．これらについて，図 2 の中に示すように，各機能に (a) 〜 (e) の表記与え，これを用いて順に解説する．ただし，複数の追加機能に関係のある Isosurfaces() 関数における変更点についてはここで示すことにする．

そもそも，Isosurfaces() 関数は Lobster に対する色付けやシェーディングなどを担っており，UI で変更を指示しても，ここまで変更が届かなければ意味が無い．そこで，関数の引数を増やして対処した．具体的には，.vert ファイル，.frag ファイル，Reflection Model，色に関する情報を引き渡せるように改良を加えた．これらの情報に基づき，シェーディングや Reflection については Listing 3 の 21 〜 42 行目で条件分岐及び material の指示を行っている．色情報については，cmap で参照すればいいだけの値が引き渡されるので，単純に Listing 3 の 120 〜 128 行目のようにすることで対応している．

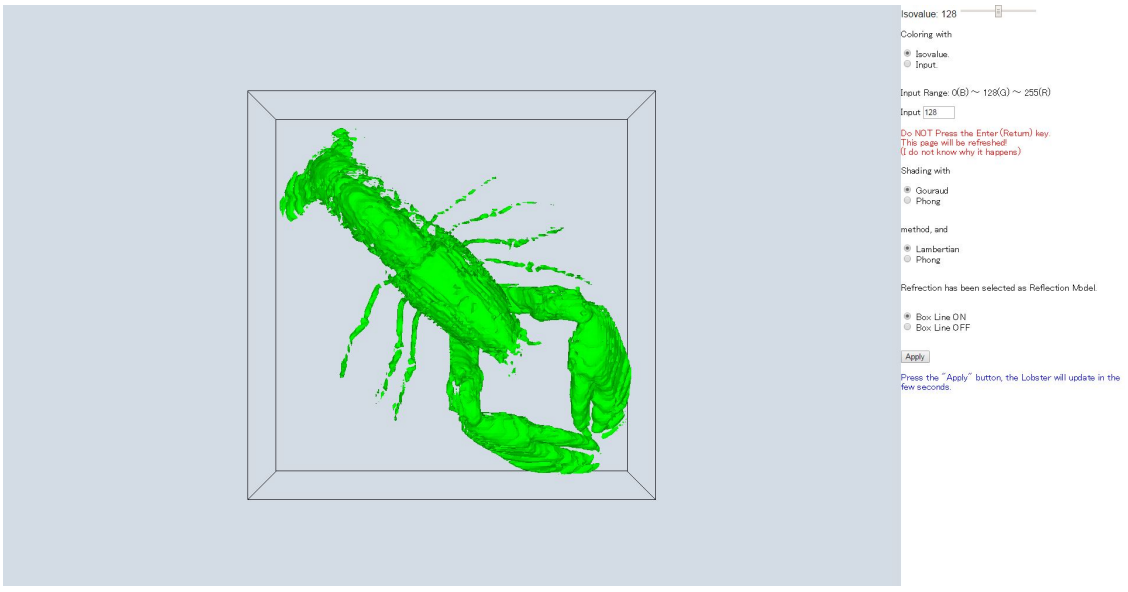

図 1　全体図

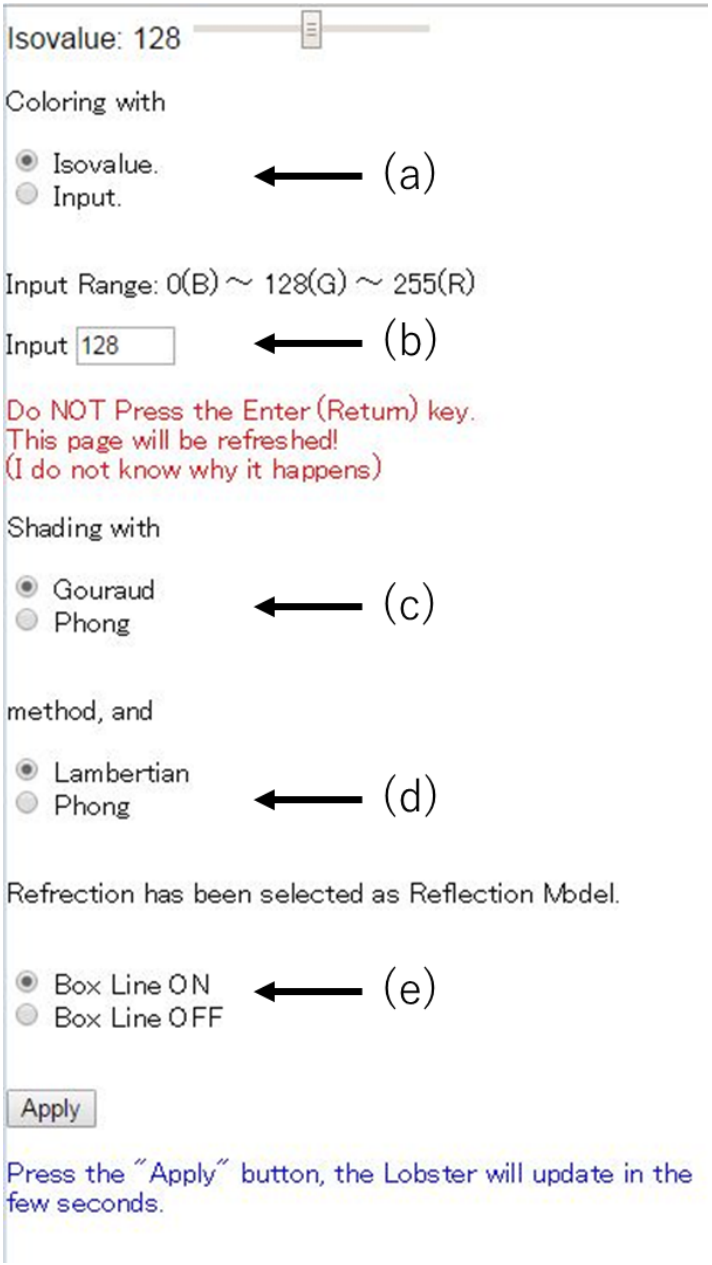

図 2　UI 拡大図

また，Apply ボタンを押した時に動作するのは，Listing 2 の 41 ～ 94 行目である．
ここからは，それぞれの機能を説明していく．

(a). ここは，UI 上にも少し示しているが，Lobster へ色をつける際の手法選択を行うための radio button である．“Isovalue” を選択すると授業中に課題や画面分割の際に例として示されていた機能を用いている．また，“Input” を選択した場合は (b) で示した場所に値を入力することで，その値に対応した色で Lobster を着色し表示することができる．具体的に，UI の表示は Listing 1 の 13 ～ 23 行目で行っている．ここは，基本的な使い方をしているので詳細は省く．一方，Apply ボタンが押された場合の (a) 部分の動作を担うソースコードは，Listing 2 (main 関数) の 83 ～ 87 行目で行っている．ここで，cmap で参照する値

を radio button の結果に基づいて指定しているので，つぎの関数 (Isosurfaces) では単純に引き受けた値の cmap 上での色を示すことになる．

(b). ここでは，(a) で示した任意の色指定を行うための値の入力を受け取るためのフォームを作成した．入力する際の目安となるように簡単に値と色の関係を示した．ここに示した値は，そのまま cmap で参照する値となっているので，特に変換を行うことなく利用できる．

(c). これは，Gouraud と Phong のどちらをシェーディングとして扱うのかを選択することができるようにするための radio button である．シェーディングを分けて考えるために，使用する変数などが異なるため，main 関数において id の名前を変えて全パターン (6 つ) 用意した．これ以降は，次に示す (d) の内容にも依存するため，ここでは説明は省略する．

(d). これは，先ほどの (c) と対象が違うだけで UI で行っていることは同様である．リフレクションモデルの選択を行うと，参照する id なども変化するので，場合分けを Listing 2 (main 関数) の 60 〜 78 行目で行っている．参照する id を組み合わせに応じて指定し，Isosurfaces 関数に渡すことで Isosurfaces 関数の変更を少なくしている．少なからず変更すべき点は出るので，そこについて少し解説すると，Isosurfaces 関数で リフレクションモデルに応じて uniform 変数の対象が変化するので，そこを場合分けしている．対応するのは，Listing 3 の 21 〜 42 行目である．このように場合分けすることで，実現することができた．

(e). 最後に, 枠線が邪魔だと感じることもあるので, 除去できるように改良した．情報の取得方法等はこれまでと同じであるが, (a) 〜 (d) とは異なり, main 関数内で解決できる内容なので, 新たに別の関数に引き渡す必要がなく，Listing 2 の 48 〜 58 行目だけで動作可能である．内容としては，上書き動作がうまくいかないので，追加の場合は screen.scene.children[2] = bounds とすることで更新し，除去するときは ****.remove() を用いることで処理している．

なお，これらの変更を適用させるために，Lobster の再描画が必要となるが，(e) のときのように **.remove() を用いた screen.scene.remove(surfaces) がうまく動作しなかったので，直接に screen.scene.children[3] = surfaces のように上書きする形で再描画を行っている．

Listing 1　ソースコード (last_task.html)

```
<html>
    <head>
        <title>Last task</title>
    </head>
    <body style="margin:0">

        <div id="display" style="width: 80%; float: left;">
        </div>
        <div id="controller" style="width:20%; float:left;">
            <label id="label" style="font-family:Arial;"></label>
            <input type="range" min="0" max="1" step="0.01" value="0.5" id="
                isovalue" />
            <br><br>
            Coloring with
            <FORM name="color">
                <br>
                <input type="radio" name="col" checked> Isovalue.<br>
                <input type="radio" name="col"> Input.<br><br>
            </FORM>
            Input Range: 0(B) ~128(G) ~255(R) <br>
            <FORM name="input">
                <br>
                Input <input size="3" type="text" name="ic" value="128">
            </FORM>
            <p><font color="#ff0000">Do NOT Press the Enter (Return) key. <br>
                This page will be refreshed! <br> (I do not know why it happens)</font>
                </p>
            Shading with
            <FORM name="shader">
                <br>
                <input type="radio" name="shad" value="gouraud" checked>
                    Gouraud<br>
                <input type="radio" name="shad" value="phong"> Phong<br><br>
            </FORM>
            method, and
            <FORM name="ref_mod">
                <br>
                <input type="radio" name="ref" value="Lambert" checked>
                    Lambertian<br>
                <input type="radio" name="ref" value="Phong"> Phong<br><br>
            </FORM>
            Refrection has been selected as Reflection Model.<br><br>
            <FORM name="box_line">
                <br>
                <input type="radio" name="lo" value="Lambert" checked> Box Line
                    ON<br>
                <input type="radio" name="lo" value="Phong"> Box Line OFF<br>
                    <br>
            </FORM>

            <button id="change-isovalue-button" size="5" style="font-family=Arial
                ;">Apply</button>
            <p><font color="#0000ff">Press the "Apply" button, the Lobster will
                update in the few seconds.</font></p>
        </div>

```

```
48      <script src="three.min.js"></script>
49          <script src="Lut.js"></script>
50          <script src="TrackballControls.js"></script>
51          <script src="KVS.min.js"></script>
52          <script src="KVS2THREE.min.js"></script>
53          <script src="KVSLobsterData.js"></script>
54          <script src="Bounds.js"></script>
55          <script src="Isosurfaces_last.js"></script>
56      <script src="last_main.js"></script>
57
58          <script type="x-shader/x-vertex" id="gouraud_L.vert">
59              varying vec3 point_color;
60              varying vec4 point_position;
61              varying vec3 normal_vector;
62              uniform vec3 light_position;
63
64              vec3 LambertianReflection( vec3 C, vec3 L, vec3 N ){
65              float ka = 0.4;
66              float kd = 0.6;
67              float dd = max( dot( N, L ), 0.0 );
68              float Ia = ka;
69              float Id = kd * dd;
70              return C * ( Ia + Id );
71              }
72
73              void main(){
74              point_position = modelViewMatrix * vec4( position, 1.0 );
75              normal_vector = normalMatrix * normal;
76
77              vec3 C = color;
78              vec3 L = normalize( light_position - point_position.xyz );
79              vec3 N = normalize( normal_vector );
80              point_color = LambertianReflection( C, L, N );
81              gl_Position = projectionMatrix * point_position;
82              }
83          </script>
84
85          <script type="x-shader/x-vertex" id="gouraud_P.vert">
86              varying vec3 point_color;
87              varying vec4 point_position;
88              varying vec3 normal_vector;
89              uniform vec3 light_position;
90              uniform vec3 camera_position;
91
92
93              vec3 PhongReflection( vec3 C, vec3 L, vec3 N, vec3 V )
94              {
95              float ka = 0.3;
96              float kd = 0.5;
97              float ks = 0.8;
98              float n = 50.0;
99
100             vec3 R = reflect( -L, N );
101             float dd = dot( N, L );
102             float ds = 0.0;
103             if ( dd > 0.0 )
104             {
```

```
105             ds = pow( dot( R, V ), n );
106             }
107
108             float Ia = ka;
109             float Id = kd * dd;
110             float Is = ks * ds;
111             return C * ( Ia + Id + Is );
112             }
113
114
115             void main()
116             {
117             point_position = modelViewMatrix * vec4( position, 1.0 );
118             normal_vector = normalMatrix * normal;
119
120             vec3 C = color;
121             vec3 L = normalize( light_position - point_position.xyz );
122             vec3 N = normalize( normal_vector );
123             vec3 V = normalize( camera_position - point_position.xyz );
124
125             point_color = PhongReflection( C, L, N, V );
126             gl_Position = projectionMatrix * point_position;
127             }
128         </script>
129
130         <script type="x-shader/x-fragment" id="gouraud.frag">
131             varying vec3 point_color;
132
133             void main()
134             {
135             gl_FragColor = vec4( point_color, 1.0 );
136             }
137         </script>
138         <script type="x-shader/x-vertex" id="phong.vert">
139             varying vec3 point_color;
140             varying vec4 point_position;
141             varying vec3 normal_vector;
142
143             void main()
144             {
145             point_color = color;
146             point_position = modelViewMatrix * vec4( position, 1.0 );
147             normal_vector = normalMatrix * normal;
148
149             gl_Position = projectionMatrix * point_position;
150             }
151         </script>
152
153         <script type="x-shader/x-fragment" id="phong_L.frag">
154             varying vec3 point_color;
155             varying vec4 point_position;
156             varying vec3 normal_vector;
157             uniform vec3 light_position;
158             uniform vec3 camera_position;
159             uniform int reflection_model;
160
161             vec3 LambertianReflection( vec3 C, vec3 L, vec3 N )
```

```
162             {
163             float ka = 0.3;
164             float kd = 0.5;
165
166             float dd = dot( N, L );
167             float Ia = ka;
168             float Id = kd * dd;
169             return C * ( Ia + Id );
170             }
171
172             void main()
173             {
174             vec3 C = point_color;
175             vec3 L = normalize( light_position - point_position.xyz );
176             vec3 N = normalize( normal_vector );
177             vec3 V = normalize( camera_position - point_position.xyz );
178
179             vec3 shaded_color = LambertianReflection( C, L, N );
180             gl_FragColor = vec4( shaded_color, 1.0 );
181             }
182         </script>
183
184         <script type="x-shader/x-fragment" id="phong_P.frag">
185             varying vec3 point_color;
186             varying vec4 point_position;
187             varying vec3 normal_vector;
188             uniform vec3 light_position;
189             uniform vec3 camera_position;
190             uniform int reflection_model;
191
192
193             vec3 PhongReflection( vec3 C, vec3 L, vec3 N, vec3 V )
194             {
195             float ka = 0.3;
196             float kd = 0.5;
197             float ks = 0.8;
198             float n = 50.0;
199
200             vec3 R = reflect( -L, N );
201             float dd = dot( N, L );
202             float ds = 0.0;
203             if ( dd > 0.0 )
204             {
205             ds = pow( dot( R, V ), n );
206             }
207
208             float Ia = ka;
209             float Id = kd * dd;
210             float Is = ks * ds;
211             return C * ( Ia + Id + Is );
212             }
213
214             void main()
215             {
216             vec3 C = point_color;
217             vec3 L = normalize( light_position - point_position.xyz );
218             vec3 N = normalize( normal_vector );
```

```
219             vec3 V = normalize( camera_position − point_position.xyz );
220
221             vec3 shaded_color = PhongReflection( C, L, N, V );
222             gl_FragColor = vec4( shaded_color, 1.0 );
223             }
224         </script>
225
226         <script>
227             var vert_shader = document.getElementById('gouraud_L.vert').text;
228             var frag_shader = document.getElementById('gouraud.frag').text;
229             var reflection_model = "Lambert";
230             var target_dom = document.getElementById('display');
231                 main(vert_shader, frag_shader, reflection_model);
232                 //main();
233         </script>
234
235     </body>
236 </html>
```

Listing 2　ソースコード (last_main.js)

```
1  function main( vert_shader, frag_shader, reflection_model, b )
2  {
3      var volume = new KVS.LobsterData();
4      var screen = new KVS.THREEScreen();
5
6      screen.init( volume, {
7          width: window.innerWidth*0.8,
8          height: window.innerHeight,
9          targetDom: document.getElementById('display'),
10         enableAutoResize: false
11     });
12
13     setup();
14
15     function setup() {
16         var color = new KVS.Vec3(0, 0, 0);
17         var box = new KVS.BoundingBox();
18         box.setColor(color);
19         box.setWidth(2);
20
21         var smin = volume.min_value;
22         var smax = volume.max_value;
23         var isovalue = KVS.Mix(smin, smax, 0.5);
24
25         document.getElementById('label').innerHTML = "Isovalue:␣" + Math.round(
               isovalue);
26
27         var bounds = Bounds(volume);
28         screen.scene.add(bounds);
29         var surfaces = Isosurfaces(volume, Math.round(isovalue), vert_shader, frag_shader,
               reflection_model, Math.round(isovalue));
30         screen.scene.add(surfaces);
31
32         document.getElementById('isovalue')
33             .addEventListener('mousemove', function () {
34                 var value = +document.getElementById('isovalue').value;
35                 var isovalue = KVS.Mix(smin, smax, value);
```

```
36              document.getElementById('label').innerHTML = "Isovalue:␣" + Math.
                    round(isovalue);
37      });
38
39      //console.log(screen.scene)
40
41      document.getElementById('change-isovalue-button')
42          .addEventListener('click', function () {
43              var value = +document.getElementById('isovalue').value;
44              var isovalue = KVS.Mix(smin, smax, value);
45              var isosurface = new KVS.Isosurface();
46              isosurface.setIsovalue(isovalue);
47
48              if (document.box_line.lo[0].checked == true) {
49                  var b = "on";
50              } else {
51                  var b = "off";
52              }
53
54              if (b == "on") {
55                  screen.scene.children[2] = bounds;
56              } else {
57                  screen.scene.remove(bounds);
58              }
59
60              if (document.ref_mod.ref[0].checked == true) { //Lambert_ref
61                  if (document.shader.shad[0].checked == true) {
62                      var vert_name = "gouraud_L.vert";
63                      var frag_name = "gouraud.frag";
64                  }else{
65                      var vert_name = "phong.vert";
66                      var frag_name = "phong_L.frag";
67                  }
68                  var reflection_model = document.ref_mod.ref[0].value;
69              }else{ //Phong_ref
70                  if (document.shader.shad[0].checked == true) {
71                      var vert_name = "gouraud_P.vert";
72                      var frag_name = "gouraud.frag";
73                  } else {
74                      var vert_name = "phong.vert";
75                      var frag_name = "phong_P.frag";
76                  }
77                  var reflection_model = document.ref_mod.ref[1].value;
78              }
79
80              var vert_shader = document.getElementById(vert_name).text;
81              var frag_shader = document.getElementById(frag_name).text;
82
83              if (document.color.col[0].checked == true) {
84                  var cc = Math.round(isovalue);
85              } else {
86                  var cc = Math.round(document.input.ic.value);
87              }
88
89              var surfaces = new Isosurfaces( volume, Math.round(isovalue), vert_shader,
                        frag_shader, reflection_model, cc );
90              screen.scene.children[3] = surfaces;
```

```
91                  //console.log(screen.scene)
92              });
93
94      }
95
96      document.addEventListener('mousemove', function () {
97          screen.light.position.copy(screen.camera.position);
98      });
99
100     window.addEventListener('resize', function () {
101         screen.resize([window.innerWidth * 0.8, window.innerHeight]);
102     });
103
104     screen.loop();
105 }
```

Listing 3　ソースコード (Isosurfaces_last.js)

```
1  function Isosurfaces(volume, isovalue, vert_shader, frag_shader, reflection_model, cc)
2  {
3      var width = 250;
4      var height = 250;
5
6      var scene = new THREE.Scene();
7      var light = new THREE.PointLight();
8      light.position.set( 0, 0, 5 );
9      scene.add(light);
10     var fov = 45;
11     var aspect = width / height;
12     var near = 1;
13     var far = 1000;
14     var camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
15     camera.position.set(0, 0, 5);
16     scene.add(camera);
17
18     var geometry = new THREE.Geometry();
19     //var material = new THREE.MeshLambertMaterial();
20
21     if (reflection_model == "Lambert") {
22         var material = new THREE.ShaderMaterial({
23             vertexColors: THREE.VertexColors,
24             vertexShader: vert_shader,
25             fragmentShader: frag_shader,
26             uniforms: {
27                 light_position:{type:'v3', value: light.position }
28             }
29         });
30     }
31
32     if (reflection_model == "Phong") {
33         var material = new THREE.ShaderMaterial({
34             vertexColors: THREE.VertexColors,
35             vertexShader: vert_shader,
36             fragmentShader: frag_shader,
37             uniforms: {
38                 light_position:{ type: 'v3', value: light.position },
39                 camera_position: { type: 'v3', value: camera.position },
40             }
```

```
41          });
42      }
43
44      console.log(material)
45
46      var smin = volume.min_value;
47      var smax = volume.max_value;
48      isovalue = KVS.Clamp( isovalue, smin, smax );
49
50      var lut = new KVS.MarchingCubesTable();
51      var cell_index = 0;
52      var counter = 0;
53      for ( var z = 0; z < volume.resolution.z − 1; z++ )
54      {
55          for ( var y = 0; y < volume.resolution.y − 1; y++ )
56          {
57              for ( var x = 0; x < volume.resolution.x − 1; x++ )
58              {
59                  var indices = cell_node_indices( cell_index++ );
60                  var index = table_index( indices );
61                  if ( index == 0 ) { continue; }
62                  if ( index == 255 ) { continue; }
63
64                  for ( var j = 0; lut.edgeID[index][j] != −1; j += 3 )
65                  {
66                      var eid0 = lut.edgeID[index][j];
67                      var eid1 = lut.edgeID[index][j+2];
68                      var eid2 = lut.edgeID[index][j+1];
69
70                      var vid0 = lut.vertexID[eid0][0];
71                      var vid1 = lut.vertexID[eid0][1];
72                      var vid2 = lut.vertexID[eid1][0];
73                      var vid3 = lut.vertexID[eid1][1];
74                      var vid4 = lut.vertexID[eid2][0];
75                      var vid5 = lut.vertexID[eid2][1];
76
77                      var v0 = new THREE.Vector3( x + vid0[0], y + vid0[1], z + vid0[2] );
78                      var v1 = new THREE.Vector3( x + vid1[0], y + vid1[1], z + vid1[2] );
79                      var v2 = new THREE.Vector3( x + vid2[0], y + vid2[1], z + vid2[2] );
80                      var v3 = new THREE.Vector3( x + vid3[0], y + vid3[1], z + vid3[2] );
81                      var v4 = new THREE.Vector3( x + vid4[0], y + vid4[1], z + vid4[2] );
82                      var v5 = new THREE.Vector3( x + vid5[0], y + vid5[1], z + vid5[2] );
83
84                      var v01 = interpolated_vertex( v0, v1, isovalue );
85                      var v23 = interpolated_vertex( v2, v3, isovalue );
86                      var v45 = interpolated_vertex( v4, v5, isovalue );
87
88                      geometry.vertices.push( v01 );
89                      geometry.vertices.push( v23 );
90                      geometry.vertices.push( v45 );
91
92                      var id0 = counter++;
93                      var id1 = counter++;
94                      var id2 = counter++;
95                      geometry.faces.push( new THREE.Face3( id0, id1, id2 ) );
96                  }
97              }
```

```
98              cell_index++;
99          }
100         cell_index += volume.resolution.x;
101     }
102
103     geometry.computeVertexNormals();
104
105     // Create color map
106     var cmap = [];
107
108     for ( var i = 0; i < 256 ; i++ )
109     {
110         var S = i/255.0; // [0,1]
111         var R = Math.max( Math.cos( ( S - 1.0 ) * Math.PI ), 0.0 );
112         var G = Math.max( Math.cos( ( S - 0.5 ) * Math.PI ), 0.0 );
113         var B = Math.max( Math.cos( S * Math.PI ), 0.0 );
114         var color = new THREE.Color( R, G, B );
115
116         cmap.push( [ S, '0x' + color.getHexString() ] );
117     }
118
119     material.vertexColors = THREE.VertexColors;
120     for ( var i = 0; i < geometry.faces.length; i++ )
121     {
122         var C0 = new THREE.Color().setHex( cmap[ cc ][1] );
123         var C1 = new THREE.Color().setHex( cmap[ cc ][1] );
124         var C2 = new THREE.Color().setHex( cmap[ cc ][1] );
125         geometry.faces[i].vertexColors.push( C0 );
126         geometry.faces[i].vertexColors.push( C1 );
127         geometry.faces[i].vertexColors.push( C2 );
128     }
129
130     return new THREE.Mesh( geometry, material );
131
132     function cell_node_indices( cell_index )
133     {
134         var lines = volume.resolution.x;
135         var slices = volume.resolution.x * volume.resolution.y;
136
137         var id0 = cell_index;
138         var id1 = id0 + 1;
139         var id2 = id1 + lines;
140         var id3 = id0 + lines;
141         var id4 = id0 + slices;
142         var id5 = id1 + slices;
143         var id6 = id2 + slices;
144         var id7 = id3 + slices;
145
146         return [ id0, id1, id2, id3, id4, id5, id6, id7 ];
147     }
148
149     function table_index( indices )
150     {
151         var s0 = volume.values[ indices[0] ][0];
152         var s1 = volume.values[ indices[1] ][0];
153         var s2 = volume.values[ indices[2] ][0];
154         var s3 = volume.values[ indices[3] ][0];
```

```
155             var s4 = volume.values[ indices[4] ][0];
156             var s5 = volume.values[ indices[5] ][0];
157             var s6 = volume.values[ indices[6] ][0];
158             var s7 = volume.values[ indices[7] ][0];
159
160             var index = 0;
161             if ( s0 > isovalue ) { index |= 1; }
162             if ( s1 > isovalue ) { index |= 2; }
163             if ( s2 > isovalue ) { index |= 4; }
164             if ( s3 > isovalue ) { index |= 8; }
165             if ( s4 > isovalue ) { index |= 16; }
166             if ( s5 > isovalue ) { index |= 32; }
167             if ( s6 > isovalue ) { index |= 64; }
168             if ( s7 > isovalue ) { index |= 128; }
169
170             return index;
171         }
172
173         function interpolated_vertex( v0, v1, s )
174         {
175             var lines = volume.resolution.x;
176             var slices = volume.resolution.x * volume.resolution.y;
177             var id_v0 = v0.x + lines * v0.y + slices * v0.z;
178             var id_v1 = v1.x + lines * v1.y + slices * v1.z;
179             var s0 = volume.values[ id_v0 ][0];
180             var s1 = volume.values[ id_v1 ][0];
181             var t = (s - s0)/(s1 - s0);
182             return new THREE.Vector3().lerpVectors( v0, v1, t );
183         }
184 }
```