Gasket1:

```
var u = add( vertices[0], vertices[1] );
var v = add( vertices[0], vertices[2] );
var p = scale( 0.25, add( u, v ) );
```

Cria um ponto P inicial entre os vertices 0, 1 e 2.

```
points = [ p ];

// Compute new points
// Each new point is located midway between
// last point and a randomly chosen vertex

for ( var i = 0; points.length < NumPoints; ++i ) {
    var j = Math.floor(Math.random() * 3);
    p = add( points[i], vertices[j] );
    p = scale( 0.5, p );
    points.push( p );
}
```

Adiciona o ponto P inicial num vetor, de acordo com o NumPoints dado no início do código (neste exemplo 5000), em cada iteração é escolhido um número aleatório j, no ponto médio entre points[i] e vertices[j] é adicionado um novo ponto ao vetor points, esse processo é repetido até terem 5000 pontos.

Gasket2:

```
function triangle( a, b, c )
{
    points.push( a, b, c );
}

function divideTriangle( a, b, c, count )
{

    // check for end of recursion

    if ( count === 0 ) {
        triangle( a, b, c );
    }
    else {

        //bisect the sides

        var ab = mix( a, b, 0.5 );
        var ac = mix( a, c, 0.5 );
        var bc = mix( b, c, 0.5 );

        --count;

        // three new triangles

        divideTriangle( a, ab, ac, count );
        divideTriangle( c, ac, bc, count );
        divideTriangle( b, bc, ab, count );
    }
}

function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLES, 0, points.length );
}
```

Função principal do código, dado 3 vértices e o número de vezes que eles têm que ser subdivididos, será pego o ponto médio dos 3 lados do triangulo formado pelos vértices, e será repetido o processo para os triângulos formados pelos vértices iniciais e os novos pontos médios até que a variável count seja 0, no final da recursão será adicionado no vetor points os vértices de todos os triângulos a serem desenhados.

Gasket3:

```
21
22      var vertices = [
23          vec3( -0.5, -0.5, -0.5 ),
24          vec3(  0.5, -0.5, -0.5 ),
25          vec3(  0.0,  0.5,  0.0 ),
26          vec3(  0.0, -0.5, 0.5 ),
27      ];
28
29
30      points = [ vec3( 0.0, 0.0, 0.0 ) ];
31
32      for ( var i = 0; points.length < NumPoints; ++i ) {
33          var j = Math.floor(Math.random() * 4);
34
35          points.push(mix(points[i], vertices[j], 0.5) );
36      }
```

Mesmo raciocínio de Gasket1, só que adicionando 1 dimensão a mais (como Gasket1 era 2d este será 3d), necessitando assim 4 vertices com posições x, y, z para formar uma pirâmide.

Gasket4:

```
72
73  function triangle( a, b, c, color )
74  {
75
76      // add colors and vertices for one triangle
77
78      var baseColors = [
79          vec3(1.0, 0.0, 0.0),
80          vec3(0.0, 1.0, 0.0),
81          vec3(0.0, 0.0, 1.0),
82          vec3(0.0, 0.0, 0.0)
83      ];
84
85      colors.push( baseColors[color] );
86      points.push( a );
87      colors.push( baseColors[color] );
88      points.push( b );
89      colors.push( baseColors[color] );
90      points.push( c );
91  }
92
93  function tetra( a, b, c, d )
94  {
95      // tetrahedron with each side using
96      // a different color
97
98      triangle( a, c, b, 0 );
99      triangle( a, c, d, 1 );
100     triangle( a, b, d, 2 );
101     triangle( b, c, d, 3 );
102 }
103
```

Mesmo raciocínio de Gasket3, só que subdivide tetraedros ao invés de triângulos, pintando cada lado do tetraedro com a função triangle, de vermelho, verde e azul.

Gasket5:

```
recursive steps  0 <input id="slider" type="range"
 min="0" max="5" step="1" value="0" />
 5
```

Mesmo raciocínio de Gasket3 porém a quantidade de vezes que o triangulo será subdividido é controlado por um slider presente no arquivo html.

```
    document.getElementById("slider").onchange = function(event) {
    numTimesToSubdivide = event.target.value;
    render();
```

Parte do código em que a variável numTimesToSubdivide recebe o valor atual do slider.