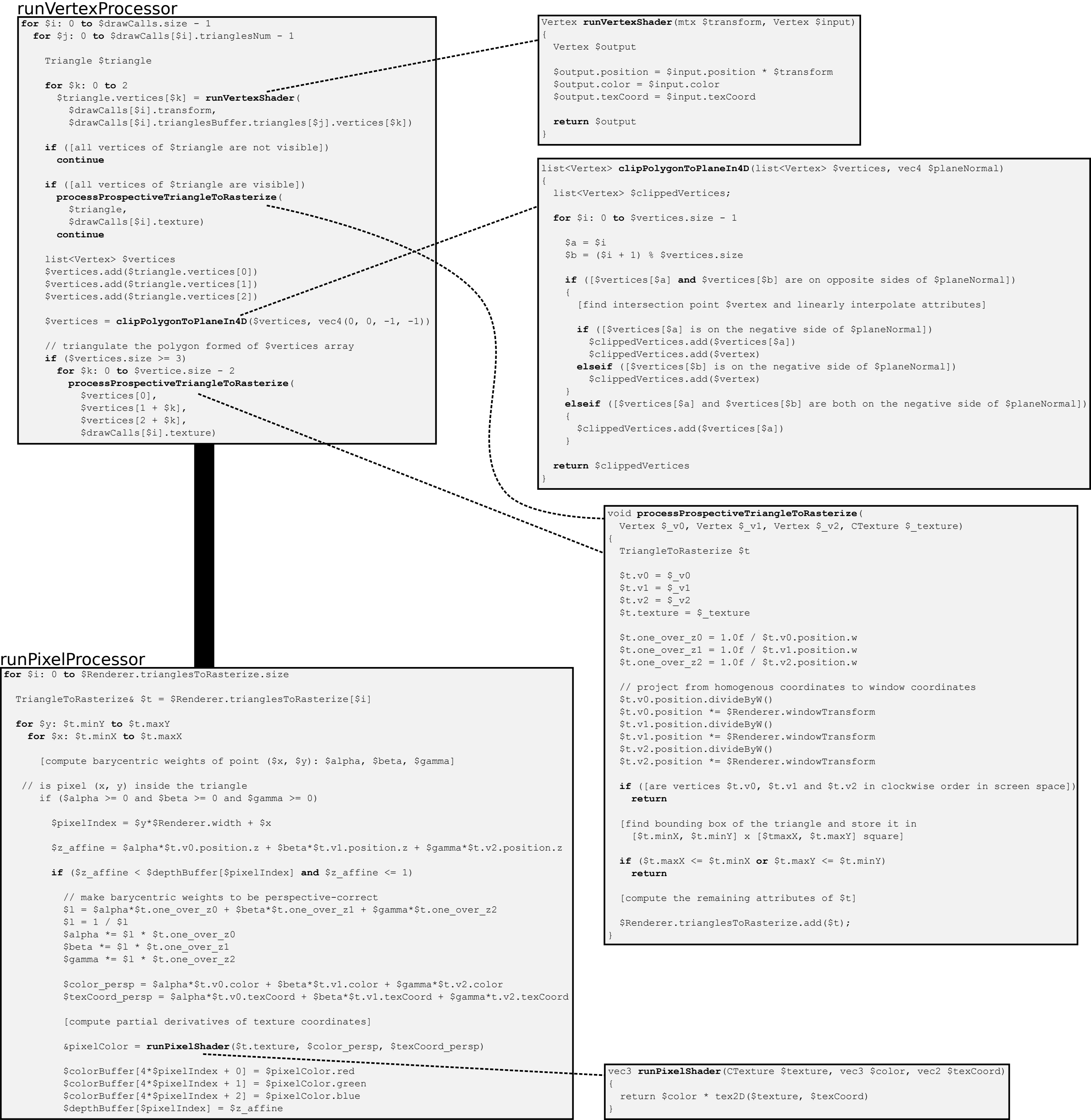

```
for $i: 0 to $drawCalls.size - 1
  for $j: 0 to $drawCalls[$i].trianglesNum - 1

    Triangle $triangle

    for $k: 0 to 2
      $triangle.vertices[$k] = runVertexShader(
        $drawCalls[$i].transform,
        $drawCalls[$i].trianglesBuffer.triangles[$j].vertices[$k])

    if ([all vertices of $triangle are not visible])
      continue

    if ([all vertices of $triangle are visible])
      processProspectiveTriangleToRasterize(
        $triangle,
        $drawCalls[$i].texture)
      continue

    list<Vertex> $vertices
    $vertices.add($triangle.vertices[0])
    $vertices.add($triangle.vertices[1])
    $vertices.add($triangle.vertices[2])

    $vertices = clipPolygonToPlaneIn4D($vertices, vec4(0, 0, -1, -1))

    // triangulate the polygon formed of $vertices array
    if ($vertices.size >= 3)
      for $k: 0 to $vertice.size - 2
        processProspectiveTriangleToRasterize(
          $vertices[0],
          $vertices[1 + $k],
          $vertices[2 + $k],
          $drawCalls[$i].texture)
```

```
Vertex runVertexShader(mtx $transform, Vertex $input)
{
  Vertex $output

  $output.position = $input.position * $transform
  $output.color = $input.color
  $output.texCoord = $input.texCoord

  return $output
}
```

```
list<Vertex> clipPolygonToPlaneIn4D(list<Vertex> $vertices, vec4 $planeNormal)
{
  list<Vertex> $clippedVertices;

  for $i: 0 to $vertices.size - 1

    $a = $i
    $b = ($i + 1) % $vertices.size

    if ([$vertices[$a] and $vertices[$b] are on opposite sides of $planeNormal])
    {
      [find intersection point $vertex and linearly interpolate attributes]

      if ([$vertices[$a] is on the negative side of $planeNormal])
        $clippedVertices.add($vertices[$a])
        $clippedVertices.add($vertex)
      elseif ([$vertices[$b] is on the negative side of $planeNormal])
        $clippedVertices.add($vertex)
    }
    elseif ([$vertices[$a] and $vertices[$b] are both on the negative side of $planeNormal])
    {
      $clippedVertices.add($vertices[$a])
    }

  return $clippedVertices
}
```

```
void processProspectiveTriangleToRasterize(
  Vertex $_v0, Vertex $_v1, Vertex $_v2, CTexture $_texture)
{
  TriangleToRasterize $t

  $t.v0 = $_v0
  $t.v1 = $_v1
  $t.v2 = $_v2
  $t.texture = $_texture

  $t.one_over_z0 = 1.0f / $t.v0.position.w
  $t.one_over_z1 = 1.0f / $t.v1.position.w
  $t.one_over_z2 = 1.0f / $t.v2.position.w

  // project from homogenous coordinates to window coordinates
  $t.v0.position.divideByW()
  $t.v0.position *= $Renderer.windowTransform
  $t.v1.position.divideByW()
  $t.v1.position *= $Renderer.windowTransform
  $t.v2.position.divideByW()
  $t.v2.position *= $Renderer.windowTransform

  if ([are vertices $t.v0, $t.v1 and $t.v2 in clockwise order in screen space])
    return

  [find bounding box of the triangle and store it in
    [$t.minX, $t.minY] x [$tmaxX, $t.maxY] square]

  if ($t.maxX <= $t.minX or $t.maxY <= $t.minY)
    return

  [compute the remaining attributes of $t]

  $Renderer.trianglesToRasterize.add($t);
}
```

```
for $i: 0 to $Renderer.trianglesToRasterize.size

  TriangleToRasterize& $t = $Renderer.trianglesToRasterize[$i]

  for $y: $t.minY to $t.maxY
    for $x: $t.minX to $t.maxX

      [compute barycentric weights of point ($x, $y): $alpha, $beta, $gamma]

   // is pixel (x, y) inside the triangle
      if ($alpha >= 0 and $beta >= 0 and $gamma >= 0)

        $pixelIndex = $y*$Renderer.width + $x

        $z_affine = $alpha*$t.v0.position.z + $beta*$t.v1.position.z + $gamma*$t.v2.position.z

        if ($z_affine < $depthBuffer[$pixelIndex] and $z_affine <= 1)

          // make barycentric weights to be perspective-correct
          $l = $alpha*$t.one_over_z0 + $beta*$t.one_over_z1 + $gamma*$t.one_over_z2
          $l = 1 / $l
          $alpha *= $l * $t.one_over_z0
          $beta *= $l * $t.one_over_z1
          $gamma *= $l * $t.one_over_z2

          $color_persp = $alpha*$t.v0.color + $beta*$t.v1.color + $gamma*$t.v2.color
          $texCoord_persp = $alpha*$t.v0.texCoord + $beta*$t.v1.texCoord + $gamma*t.v2.texCoord

          [compute partial derivatives of texture coordinates]

          &pixelColor = runPixelShader($t.texture, $color_persp, $texCoord_persp)

          $colorBuffer[4*$pixelIndex + 0] = $pixelColor.red
          $colorBuffer[4*$pixelIndex + 1] = $pixelColor.green
          $colorBuffer[4*$pixelIndex + 2] = $pixelColor.blue
          $depthBuffer[$pixelIndex] = $z_affine
```

```
vec3 runPixelShader(CTexture $texture, vec3 $color, vec2 $texCoord)
{
  return $color * tex2D($texture, $texCoord)
}
```