Documentation

Krisztián Szabó

August 1, 2024

In this PDF, I aim to describe concepts related to computer graphics, with a focus on the mathematical aspects. While this information may be useful to others, please note that it is primarily for my own purposes, and I do not assume responsibility for any errors.

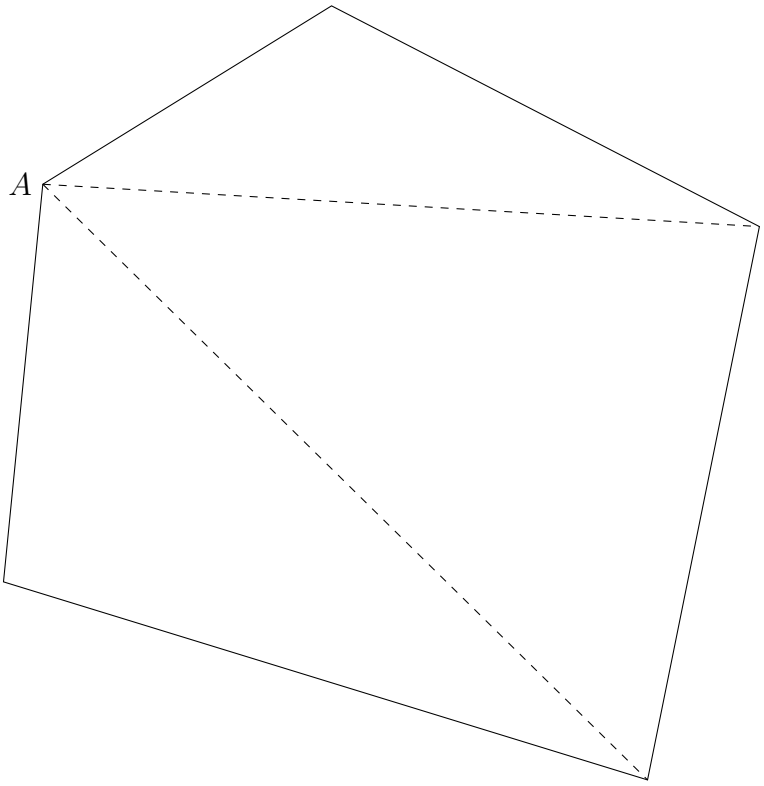
Contents

1 Convex Mesh Division

In this section, I will explain a simple yet effective method to divide any convex polyhedron into arbitrarily small tetrahedrons. This can be useful for creating destructible objects. Let us assume we have a convex polyhedron, where we only know the vertices of each face, and each face can be any polygon.

1.1 Polyhedron to Tetrahedrons

First, we aim to divide our polyhedron into smaller tetrahedrons and subsequently divide those tetrahedrons further. As you might have guessed, a tetrahedron is also a polyhedron. We will use a slightly different approach for dividing tetrahedrons. For simplicity, convert all faces into triangles, which is straightforward:

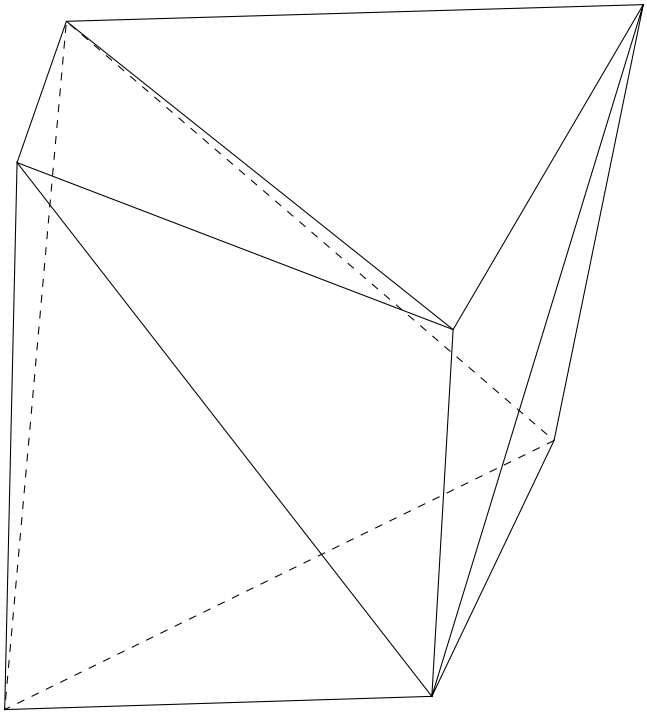


All you need to do is select any vertex A and connect it to every vertex in the polygon except its neighbors.

```
struct Triangle {
    glm::vec3 v1;
    glm::vec3 v2;
    glm::vec3 v3;
};

std::vector<Triangle> GetTrianglesFromPolygon(const std::vector<glm::vec3
    >& polygon) {
```

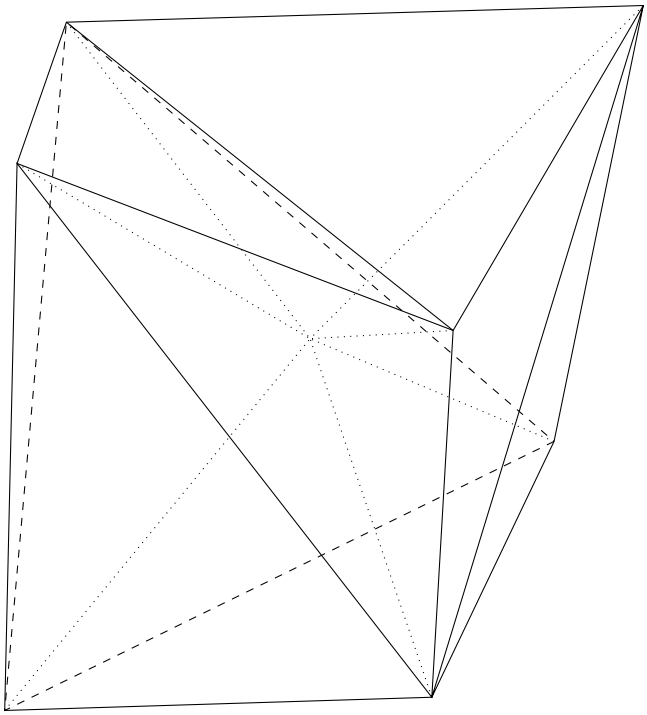
```
        std::vector<Triangle> result;

        for (int i = 2; i < polygon.size(); ++i) {
            result.push_back(Triangle{polygon[0], polygon[i-1], polygon[i]});
        }

        return result;
    }
```

Now we have any array of triangles that build up our convex polyhedron.



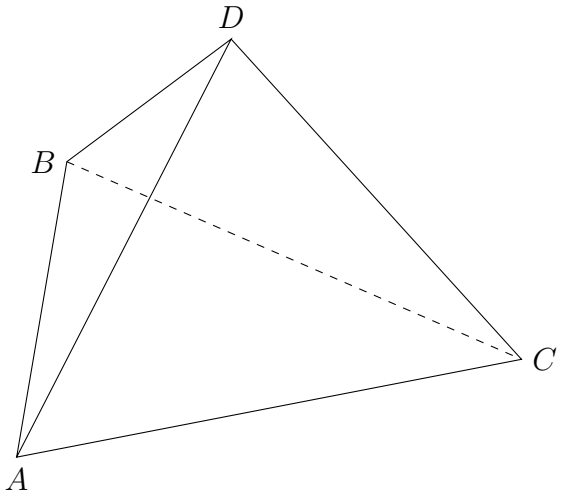
The simplest method for dividing a convex polyhedron into smaller tetrahedrons is to find the center of the polyhedron and then connect this center to each of the polyhedron's triangular faces.

```
    std::vector<Tetra> GetTetras(const std::vector<Triangle>& triangles) {
        glm::vec3 center{0.0f, 0.0f, 0.0f};
        for (const auto& t : triangles) {
            center += t.v1; center += t.v2; center += t.v3;
        }
        center /= (triangles.size() * 3.0f);
        std::vector<Tetra> tetras;
        for (const auto& t : triangles)
        {
            tetras.push_back({t.v1, t.v2, t.v3, center});
        }
        return tetras;
    }
```

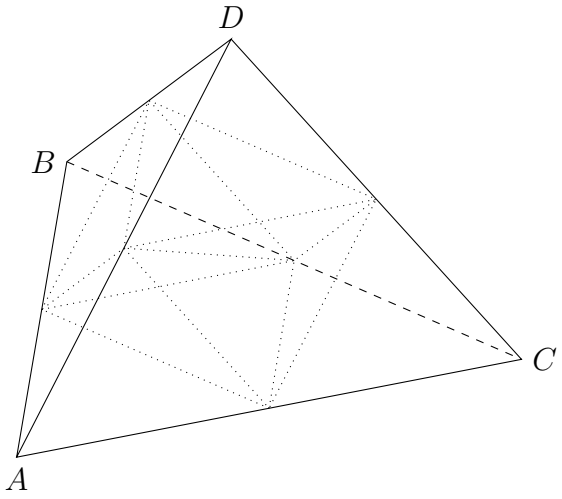


1.2 Tetrahedrons to tetrahedrons

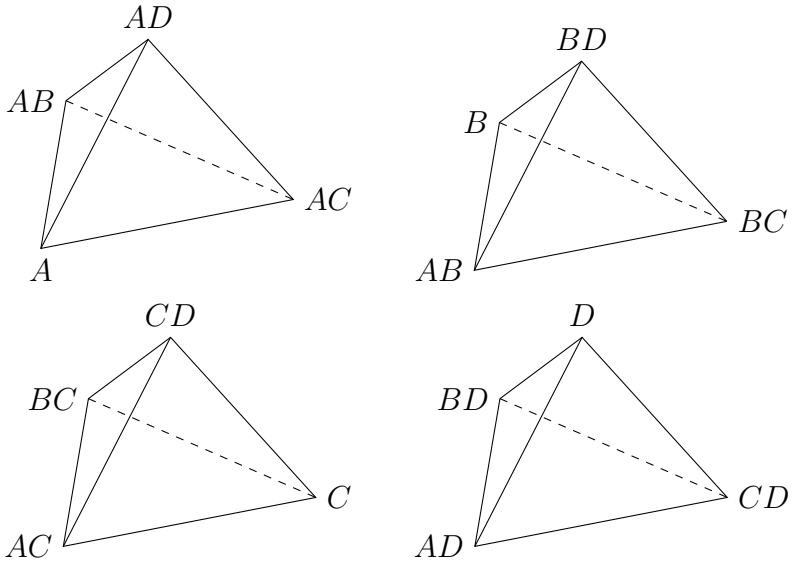
We have successfully divided our mesh into tetrahedrons. While further subdivision might be desirable, this method may not be the most effective for dividing the tetrahedrons themselves. Let's explore why this particular tetrahedron could be an example of our polyhedron:



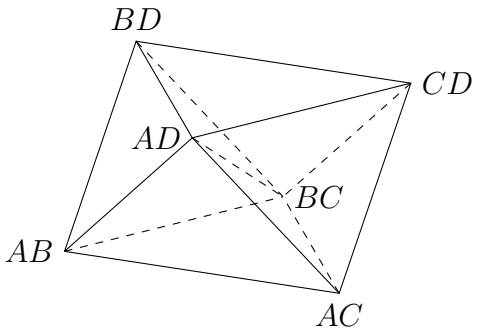
As illustrated, continuing this pattern would result in all the sub-tetrahedrons having identical faces. This would produce very thin tetrahedrons with wide faces, which might not be ideal for our purposes. Therefore, let's explore a different algorithm. Our overall goal is to calculate the midpoints of all the edges in each tetrahedron and use these midpoints to generate smaller tetrahedrons.



The illustration may be tricky, but there are actually eight smaller tetrahedrons within the larger one. Identifying the vertices is straightforward:



Within the tetrahedron, there is a hexahedron comprised of four tetrahedrons.



The algorithm is quite straightforward. You start with six vertices and need to select two that do not share a common vertex. Following this labeling approach, if you examine the labels of the chosen midpoints, they will encompass all the vertices. In this case, the appropriate pairs of vertices might be:

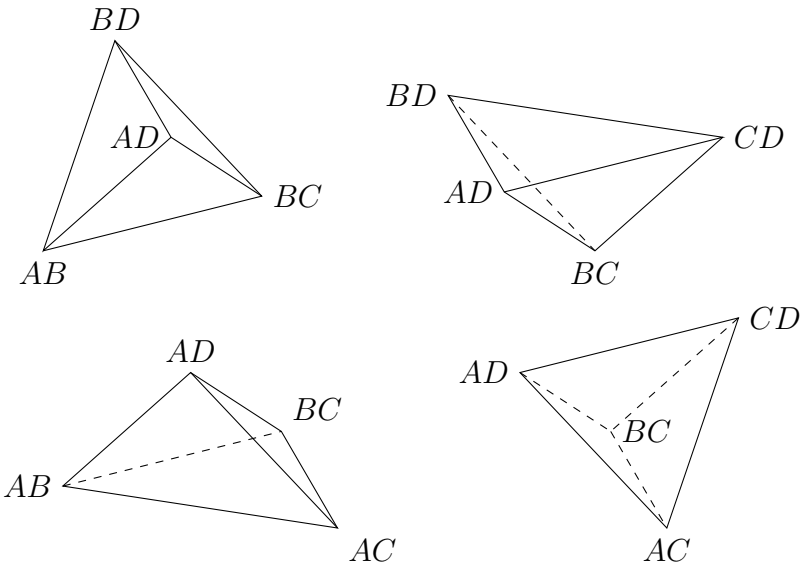
$$AD - BC,\ AB - CD,\ BD - AC$$

All four tetrahedrons will contain these two vertices. Next, we need to determine the order in which to assemble the remaining four vertices. For instance, if we choose AD and BC as the two vertices, the tetrahedrons should be assembled as follows:

1. AD, BC, $\cdot$, $\cdot$
2. AD, BC, $\cdot$, $\cdot$
3. AD, BC, $\cdot$, $\cdot$
4. AD, BC, $\cdot$, $\cdot$

where the $\cdot$ symbols represent the remaining vertices. The algorithm is also quite simple: you need to choose two vertices that share exactly one vertex.

1. AD, BC, $B\mathbf{D}$, $C\mathbf{D}$
2. AD, BC, $\mathbf{B}D$, $A\mathbf{B}$
3. AD, BC, $\mathbf{A}B$, $\mathbf{A}C$
4. AD, BC, $A\mathbf{C}$, $\mathbf{C}D$



The good news is that the midpoints calculated initially do not need to be exact; they can be any points along the edges. This flexibility can lead to a noisy division.

1.3 Triangle vertex ordering

In computer graphics, the ordering of triangle vertices is really important. When you look at your mesh at any position in the world, the triangles you are currently looking at have to be ordered either clockwise or counter clokwise. In this subsection I provide as simple algorithm to order the triangles in your mesh. Let us have a list of triangles that built up our mesh. Using the glm library here is a code:

```
bool IsClockwise(const Triangle& triangle, const glm::vec3& polyCenter)
{
    glm::vec3 triangleCenter = (triangle.v1 + triangle.v2 + triangle.v3) / 3.0f;
    glm::vec3 outVector = triangleCenter - polyCenter;

    glm::vec3 v1 = outVector.v2 - outVector.v1;
    glm::vec3 v2 = triangle.v3 - triangle.v1;
    glm::vec3 normal = glm::cross(v1, v2);

    return glm::dot(normal, outVector) > 0;
}
```

Now let us dvelve into this code with illustrations. Let we have a convex polyhedron. We can easily calculate it's centerpoint by adding up all the vertices and then dividing with the size. For simplicity I have chosen a tetrahedron: