

# Contour 구현 방법

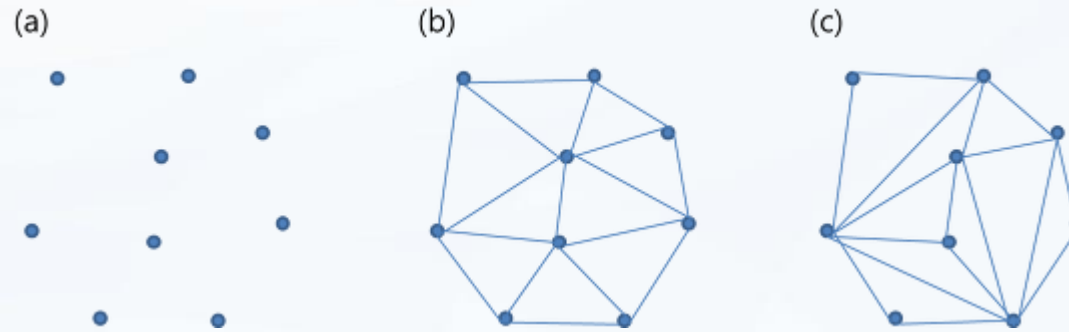
---



(주)쓰리웨이소프트

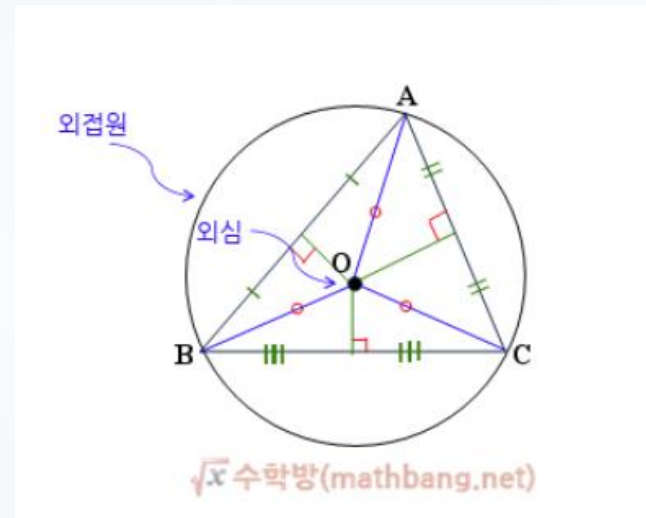
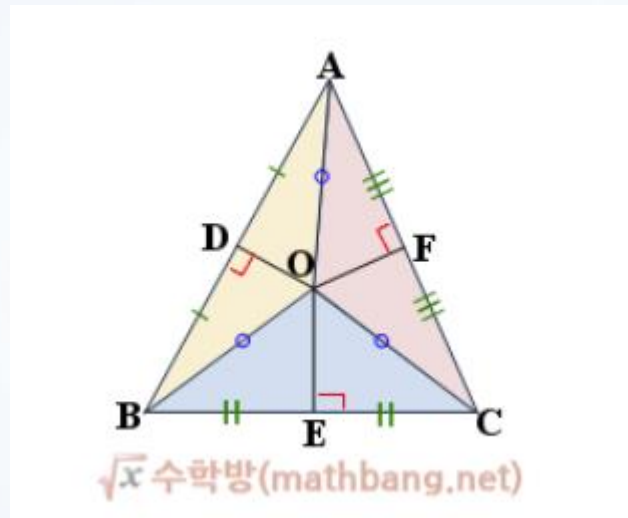
2020.08

- 들로네 삼각분할은 평면위의 점들을 삼각형으로 연결하여 공간을 분할할 때, 이 삼각형들의 내각의 최소값이 최대가 되도록 하는 분할을 말합니다.

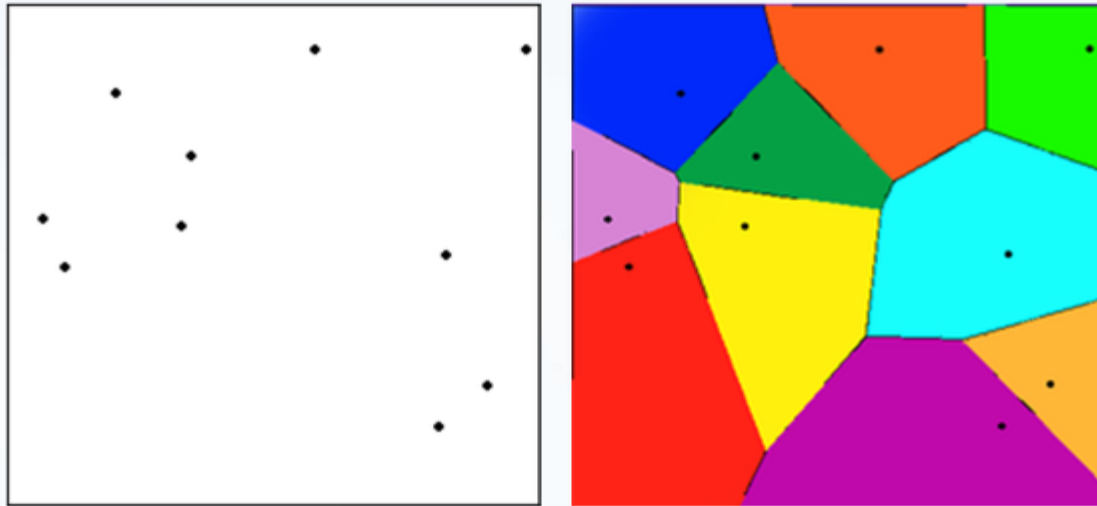


- 들로네 삼각분할은 이러한 여러 삼각분할 중에서 (b)와 같이 각각의 삼각형들이 최대한 정삼각형에 가까운 즉, (c)와 같이 길쭉하고 홀쭉한 삼각형이 나오지 않도록 하는 분할을 말합니다.
- 들로네 삼각분할의 가장 중요한 특징중 하나는, "어떤 삼각형의 외접원도 그 삼각형의 세 꼭지점을 제외한 다른 어떤 점도 포함하지 않는다" 입니다 (이를 empty circumcircle property라고 부릅니다).

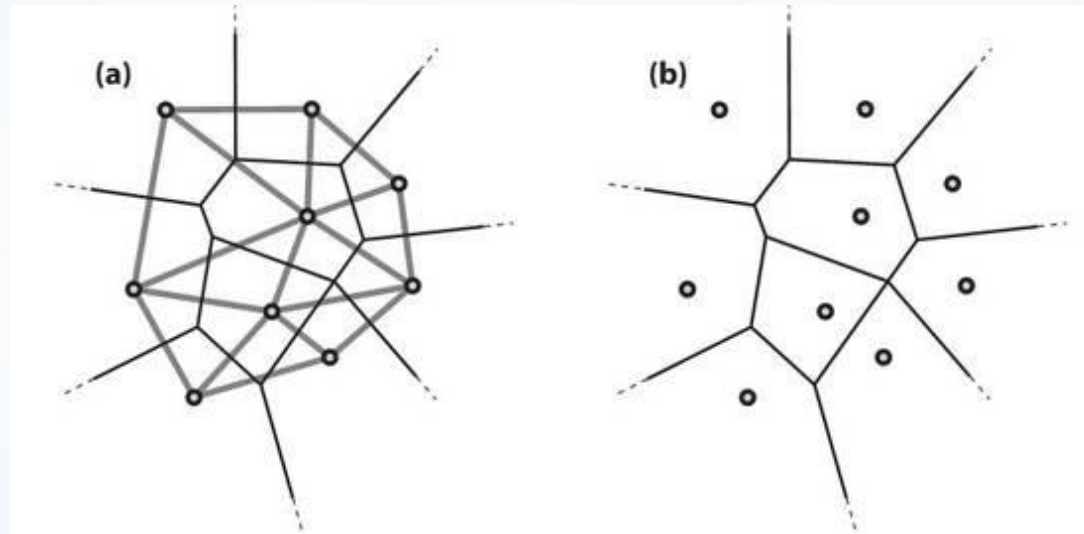
- 외심
  - 각 변의 수직 이등분선을 그으면 만나는 점
  - 각 꼭짓점에 이르는 거리가 같음
- 외접원
  - 외심을 중심으로 각꼭지점을 지나는 원



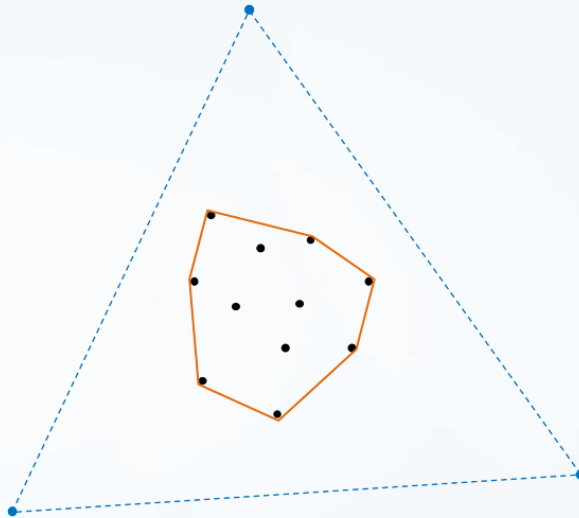
- 보로노이 다이어그램은 어떤 시드 점(seed point)들과의 거리에 따라서 평면을 분할한 그림으로서, 평면 위에 주어진 시드 점  $p_1, p_2, \dots, p_n$ 에 대한 보로노이 다이어그램은 평면 위의 점들이 어떤  $p_i$ 와 가장 가까운지에 따라서 영역을 분할한 다이어그램을 말합니다.
- 예를 들어, 아래 왼쪽 그림과 같은 사각형 공간에 10개의 시드 점 (검은색 점들)을 주면 오른쪽 그림과 같은 보로노이 다이어그램이 나옵니다.



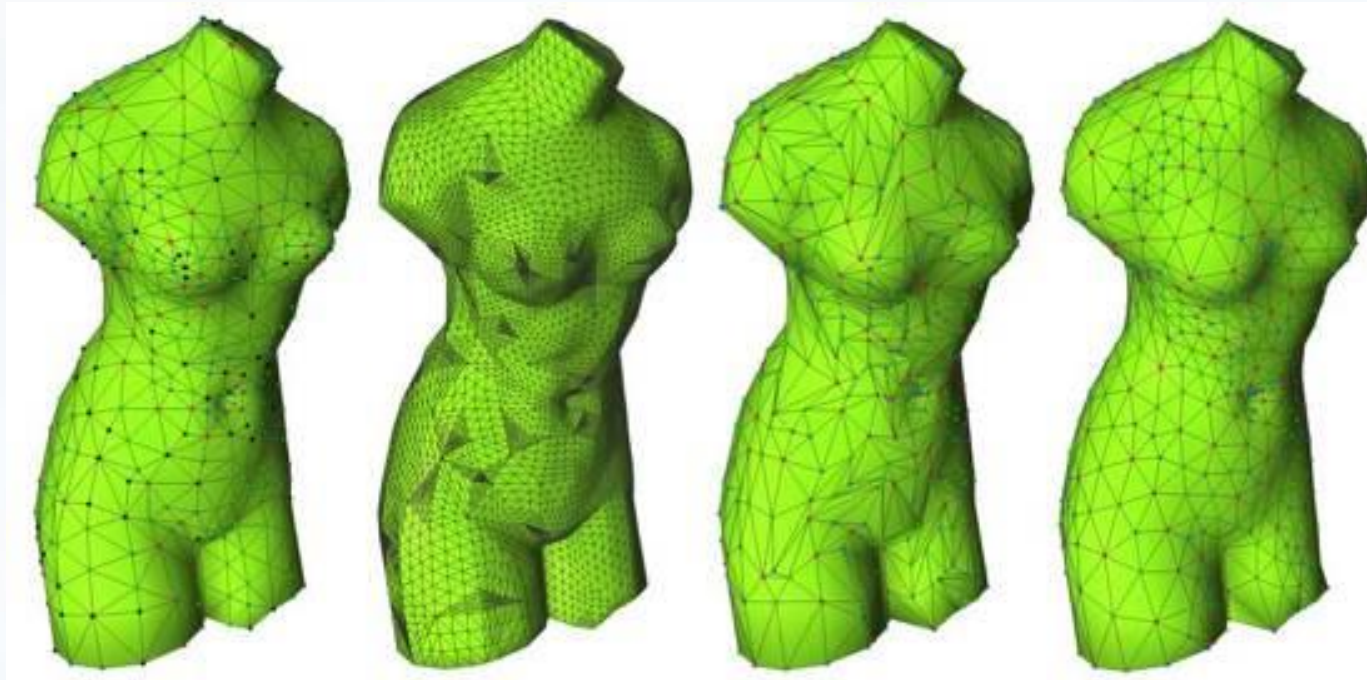
- 들로네 삼각분할과 보로노이 다이어그램은 서로 듀얼 이미지(dual image) 관계에 있다.
- [들로네->보로노이]
  - 보로노이 다이어그램의 각 seed 점들을 가지고 들로네 삼각분할을 구한 후에, 구해진 삼각형들의 외접원들의 중심을 연결하면 보로노이 다이어그램이 나옵니다.
  - 좀더 상세하게 말하면, 어떤 점을 공통의 꼭지점으로 하는 들로네 삼각형들의 외접원들의 중심을 순서대로 연결하면 이 공통점을 seed로 하는 보로노이 영역이 나옵니다.
- [보로노이->들로네]
  - 서로 인접한 보로노이 영역들간의 seed 점들을 연결하면 이 seed 점들에 대한 들로네 삼각분할이 얻어집니다.



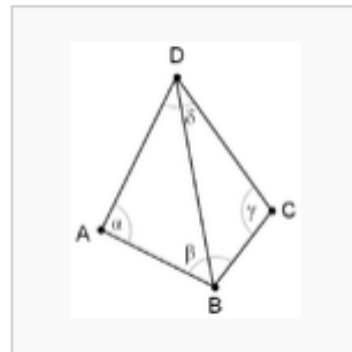
- Convex Hull 구하기
  - 어떤 점들의 집합에 대한 Convex Hull 이란 쉽게 생각하면 그 점들을 실로 팽팽하게 둘러쌀 때 생기는 볼록(convex) 다각형입니다
  - Delaunay triangulation을 이용해서 convex hull을 구하는 방법은, 원래의 점들의 집합에 외부의 점 3개를 추가하여 드로네 삼각분할을 구한 후 어떤 점들이 새로 추가된 외부의 점과 삼각형을 이루었는지를 조사하면 됩니다.



- 들로네 삼각분할의 본연(?)의 응용이라 볼 수 있습니다. 3D 입체 물체의 표면을 폴리곤으로 모델링할 때 들로네 triangulation을 이용하면 표면위의 점들의 집합으로부터 nice(?)한 폴리곤 메쉬를 얻을 수 있습니다. 3D 입체 표면을 폴리곤으로 모델링하는 이유는 빠른 랜더링(rendering)이 가능하기 때문입니다.



- 크게 incremental 알고리즘과 divide and conquer 알고리즘으로 나눌 수 있습니다.
- **Incremental 알고리즘**
  - incremental 알고리즘을 이해하기 위해서는 먼저 flipping에 대해 알아야 합니다. 어떤 사각형을 대각선을 따라 두개의 삼각형으로 나누는 방법은 2가지가 있습니다. 이 때, 만일 어느 한 분할이 Delaunay 조건을 만족하지 않는다면 다른 대각선 방향으로의 분할은 반드시 Delaunay 조건을 만족하게 됩니다. 이와 같이 분할 방향을 바꾸어서 Delaunay 조건을 만족시켜 나가는 것을 flipping이라고 합니다.



This triangulation does not meet the Delaunay condition (the sum of  $\alpha$  and  $\gamma$  is bigger than  $180^\circ$ ).



This triangulation does not meet the Delaunay condition (the circumcircles contain more than three points).

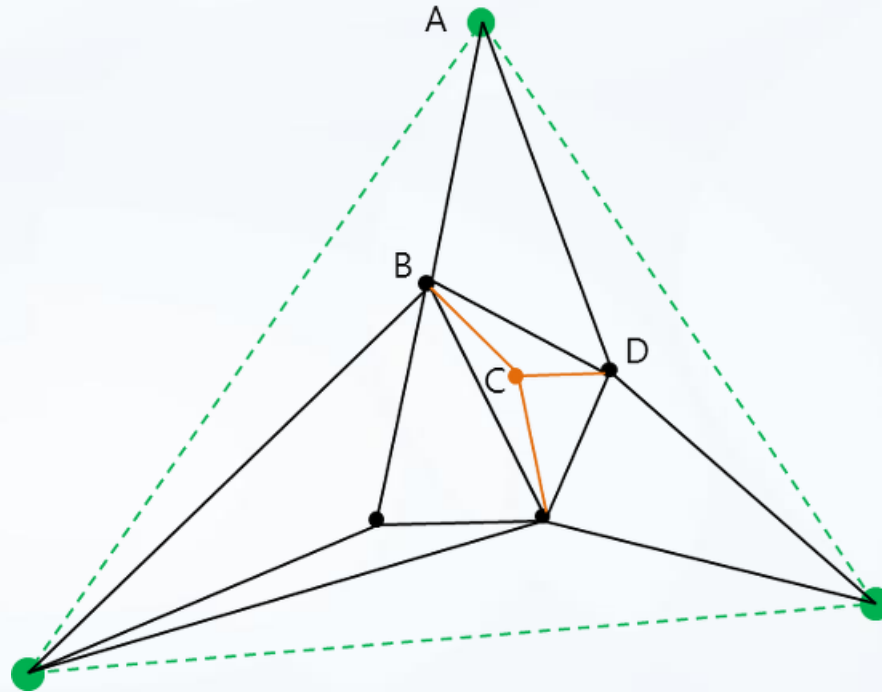


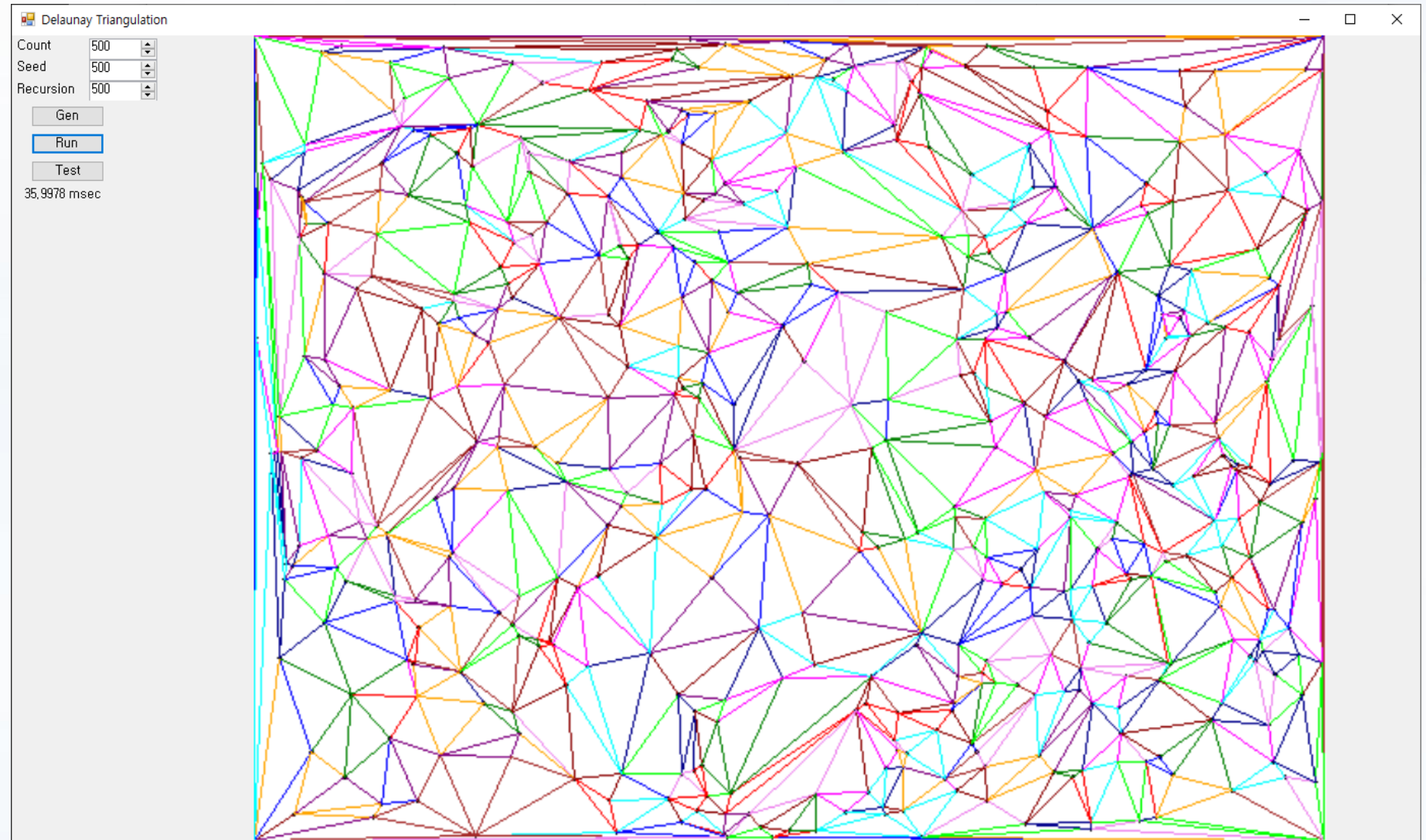
Flipping the common edge produces a Delaunay triangulation for the four points.



- **Incremental 알고리즘**

- 가장 기본적인 incremental 알고리즘은 앞서 설명한 Convex Hull 구하기 응용에서처럼 먼저 원래 점들로부터 무한히 멀리 떨어진 외부의 3 점(아래 그림의 녹색 점들)을 설정한 후에, 하나씩 원래 점들을 추가하면서 Delaunay triangulation 조건을 만족하도록 삼각분할을 조정하는 형식으로 진행합니다.



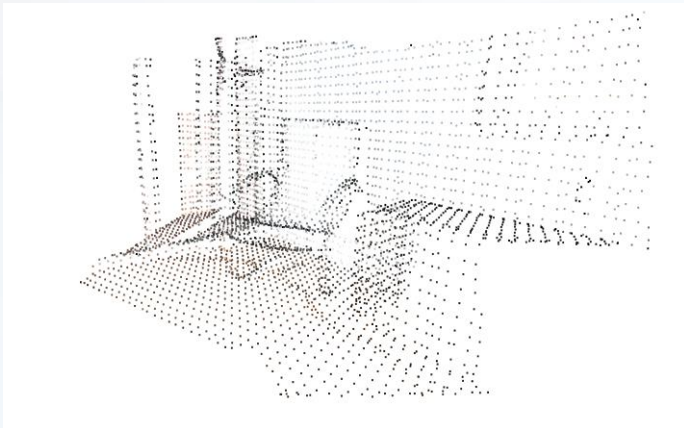


```
List<Vertex> set = new List<Vertex>();
Random.Random r = new Random.Random(seed);
for(int i = 0; i < count; i++)
{
    set.Add(new Vertex(r.Float(0,width), r.Float(0, height), 0));
}
Mesh m = new Mesh();
m.Recursion = 6;
m.Compute(set, new RectangleF(0, 0, 640, 480));
List<Vertex> p = m.Points; // Points for directX vertex buffer.

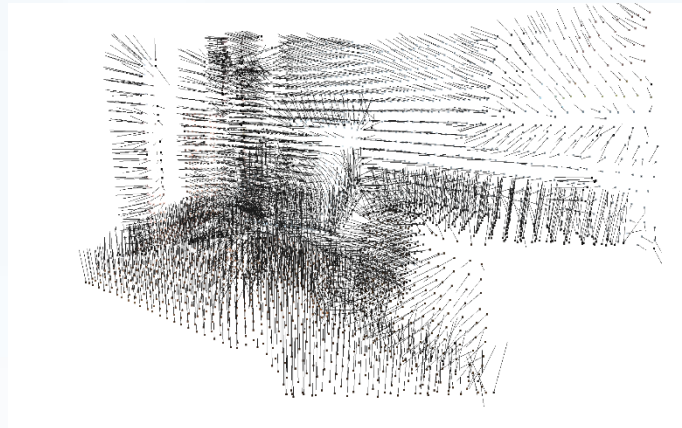
int [] indicies = m.GetVertexIndicies(); // Indexes for directX surface.
```

- <http://www.open3d.org/docs/release/tutorial/Basic/pointcloud.html#Visualize-point-cloud>

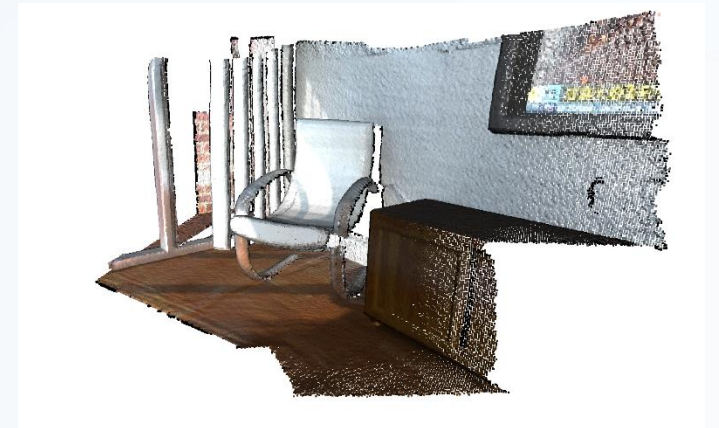
Vertex



Vertex Normal

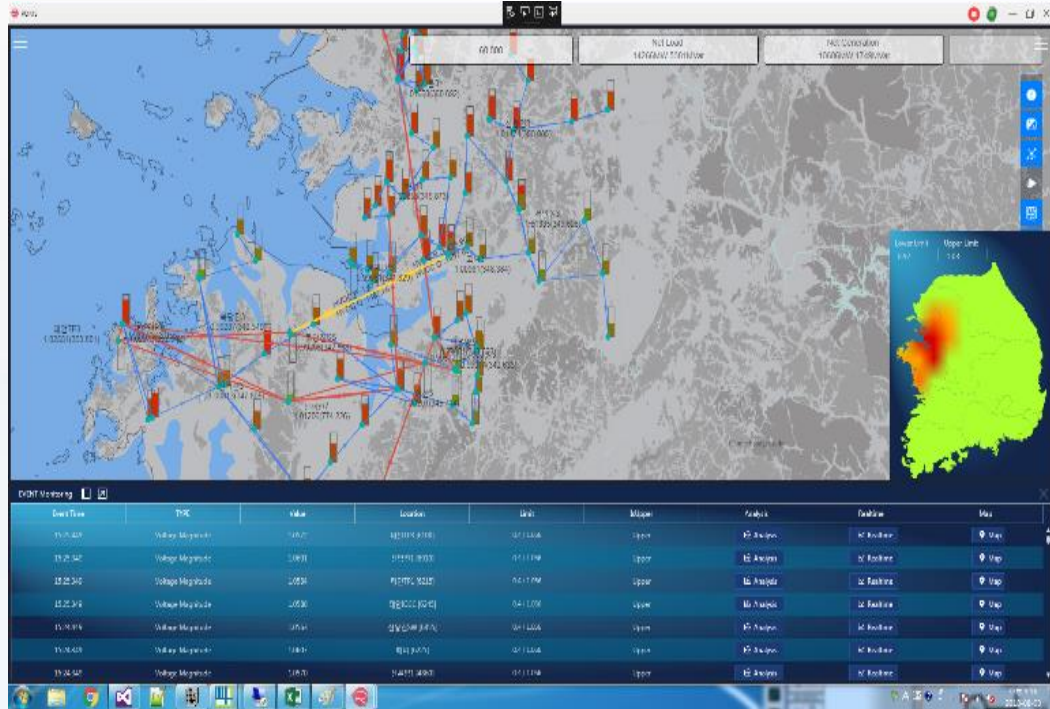


Texture/Color

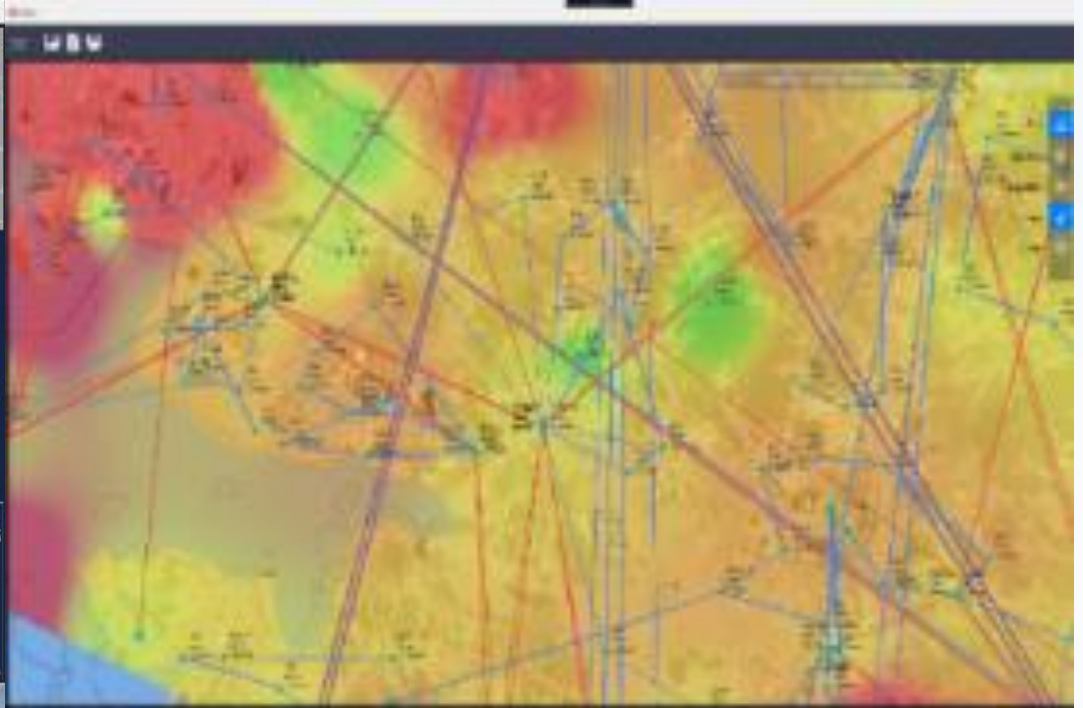


3D File (ply, obj, fbx ... )





C# Part




JavaScript Part

Bitmap Array만 만들어주면 되는 것 아닌가?


```
ConvertByteArrayToWritableBitmap
/// <summary>
/// Converts the specified image.
/// </summary>
/// <param name="image">The image.</param>
/// <returns>The WritableBitmap</returns>
public static WritableBitmap ConvertToWritableBitmap(byte[] image)
{
    var bitmapImage = new BitmapImage();
    var memoryStream = new MemoryStream(image);
    bitmapImage.SetSource(memoryStream);
    return new WritableBitmap(bitmapImage);
}
```




**KOREA**  
UNIVERSITY

chapter-7--28rasterizer-29 (2)

chapter 2 (math basics)



## Line, Ray, and Linear Interpolation (cont'd)

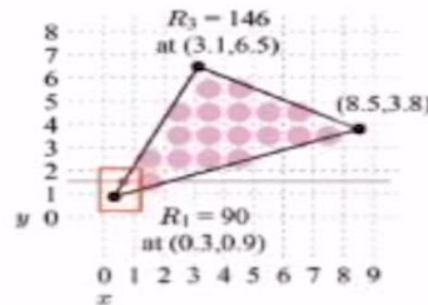
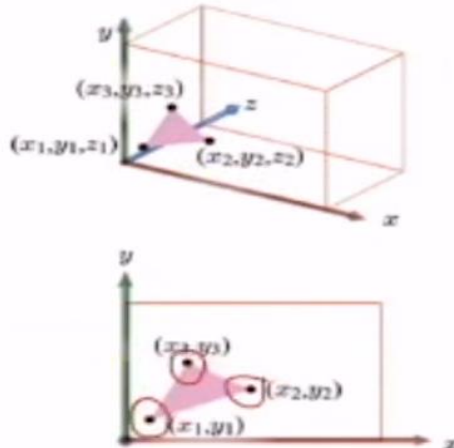
- Linear interpolation in 3D space
$$p(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} (1-t)x_0 + tx_1 \\ (1-t)y_0 + ty_1 \\ (1-t)z_0 + tz_1 \end{pmatrix}$$
- Whatever attributes are associated with the end points, they can be linearly interpolated. Suppose that the endpoints are associated with colors  $c_0$  and  $c_1$ , respectively, where  $c_0 = (R_0, G_0, B_0)$  and  $c_1 = (R_1, G_1, B_1)$ . Then, the color  $c(t)$  is defined as follows:
$$c(t) = (1-t)c_0 + tc_1 = \begin{pmatrix} (1-t)R_0 + tR_1 \\ (1-t)G_0 + tG_1 \\ (1-t)B_0 + tB_1 \end{pmatrix}$$


Introduction to Computer Graphics with OpenGL ES (J. Han)

2-13

## Scan Conversion

- Scan conversion breaks up each triangle into a set of *fragments*. It identifies the pixels covered by the triangle and *interpolates* the vertex attributes (such as normals and texture coordinates) for each pixel location.
- The per-vertex attributes usually do not include RGB color. Just for presentation, however, let's assume color attributes and use  $R$  color for scan conversion.
- A horizontal line of pixels is named a *scan line*.



$$\begin{aligned}\Delta y &= 6.5 - 0.9 = 5.6 \\ \Delta R &= 146 - 90 = 56 \\ \frac{\Delta R}{\Delta y} &= \frac{56}{5.6} = 10 \\ \Delta x &= 3.1 - 0.3 = 2.8 \\ \frac{\Delta x}{\Delta y} &= \frac{2.8}{5.6} = 0.5\end{aligned}$$

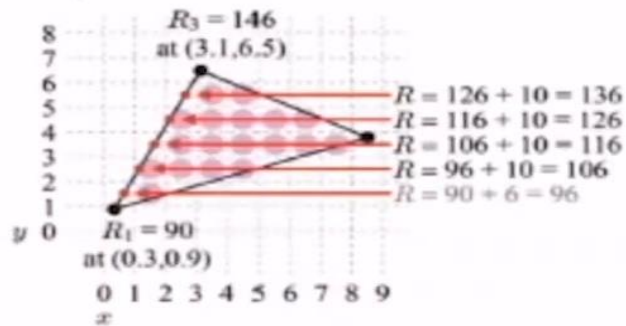
$$\begin{aligned}R &= R_1 + 0.6 \frac{\Delta R}{\Delta y} \\ &= 90 + 0.6 \times 10 = 96 \\ x &= x_1 + 0.6 \frac{\Delta x}{\Delta y} \\ &= 0.3 + 0.6 \times 0.5 = 0.6\end{aligned}$$





## Scan Conversion (cont'd)

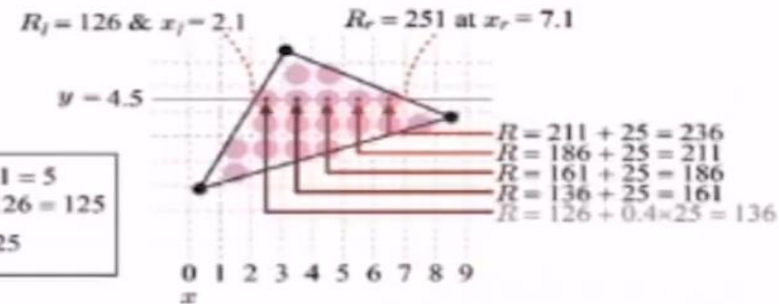
- Bilinear interpolation – along the edges first, and then along the scan lines



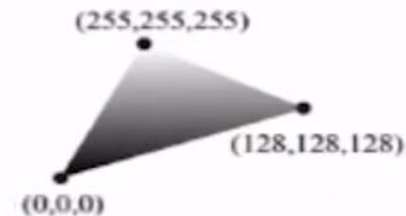
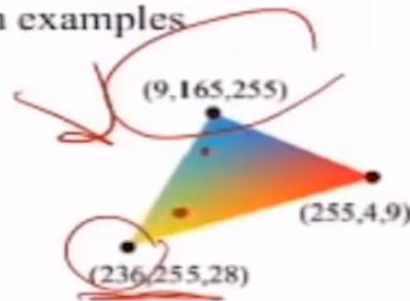
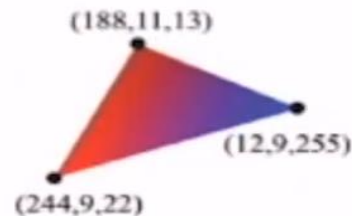
$$\Delta x = 7.1 - 2.1 = 5$$

$$\Delta R = 251 - 126 = 125$$

$$\frac{\Delta R}{\Delta x} = \frac{125}{5} = 25$$



- Color interpolation examples



# 감사합니다.

(주) 쓰리웨이소프트

ThreewaySoft 

대표이사 **최미화**

M. 010-2728-5286 T. 042-862-3330 F.042-862-3331

E. [mhchoi91@3waysoft.com](mailto:mhchoi91@3waysoft.com)

H. [www.3waysoft.com](http://www.3waysoft.com)

A. 대전광역시 유성구 문지로 193, KAIST 학부동 701호

