Computer Graphics

Prof. Jibum Kim

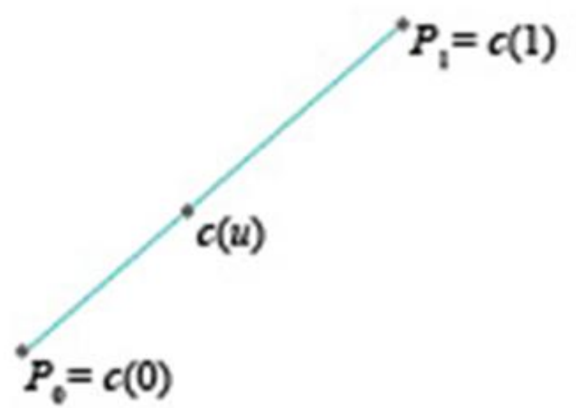
Department of Computer Science & Engineering

Incheon National University

- $\boldsymbol{B\acute{e}zier}$ Surface
- Bilinear $\boldsymbol{B\acute{e}zier}$ Patch

- 지금까지 $B\acute{e}zier$ Curve를 만드는 방법을 간단하게 배웠다
- 이를 확장하여 곡선이 아닌 직선 기반의 표면, Surface, 를 만드는 방법인 Bilinear (양방향) Bezier Patch 에 대해서 알아보자
- $B\acute{e}zier$ Curve 는 파라미터 u하나로 곡선을 만들었지만 평면 (혹은 곡면)을 만들기 위해서는 파라미터 v를 하나 더 사용한다

- Linear $Bézier$ curve
- 두 개의 control point, P0, P1만 있다고 하자
- P0와 P1을 연결하는 $Bézier$ curve (선분)는 다음과 같이 표현 가능
- $\boldsymbol{c(u) = (1-u)P_0 + uP_1}$, 단, $\boldsymbol{0 \le u \le 1}$



- **Bilinear $B\acute{e}zier$ Patch**
- **Linear $B\acute{e}zier$ curve 를 확장 한 것**
- **4개의 control points (a, b, c, d)가 주어져 있고 두 개의 파라미터 u, v 사용**

1. 선분 a 와 b 사이의 한 점: $\boldsymbol{e(u) = (1-u)a + ub,\ \ 0 \le u \le 1}$

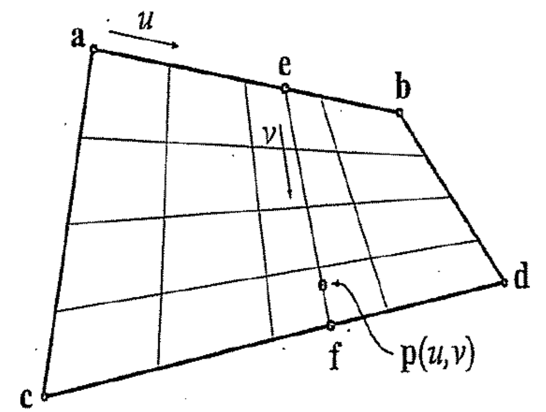
2. 선분 c 와 d 사이의 한 점: $\boldsymbol{f(u) = (1-u)c + ud,\ \ 0 \le u \le 1}$

3. 선분 e 와 f 사이의 한 점

$$\boldsymbol{p(u,v) = (1-v)e + vf, 0 \le v \le 1}$$

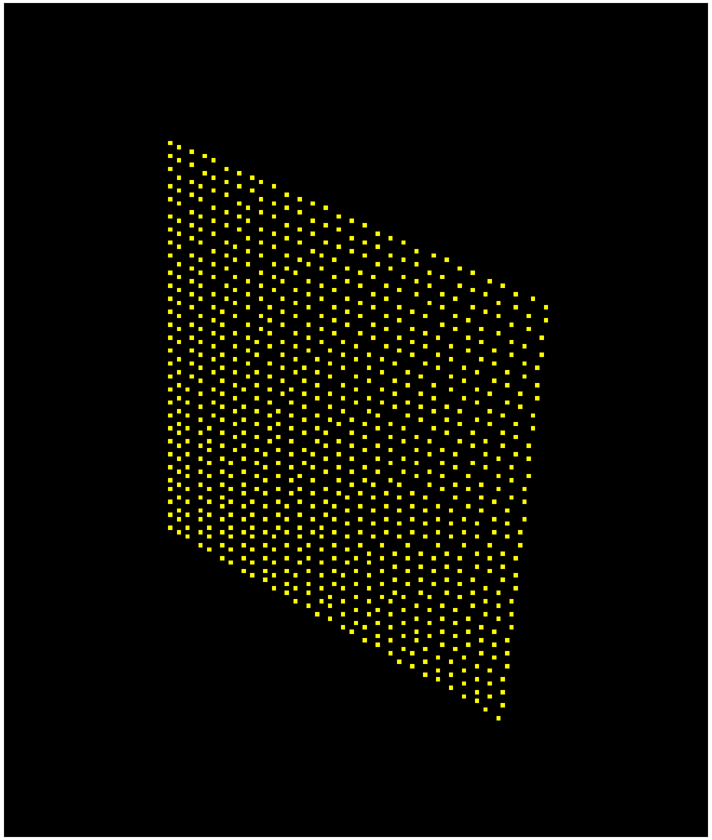
$$\boldsymbol{p(u,v) = (1-\mathrm{u})(1-\mathrm{v})\mathrm{a} + \mathrm{u}(1-\mathrm{v})\mathrm{b} + \mathrm{v}(1-\mathrm{u})\mathrm{c} + \mathrm{uvd}}$$

$$\boldsymbol{,\ \ 0 \le u \le 1, 0 \le v \le 1}$$

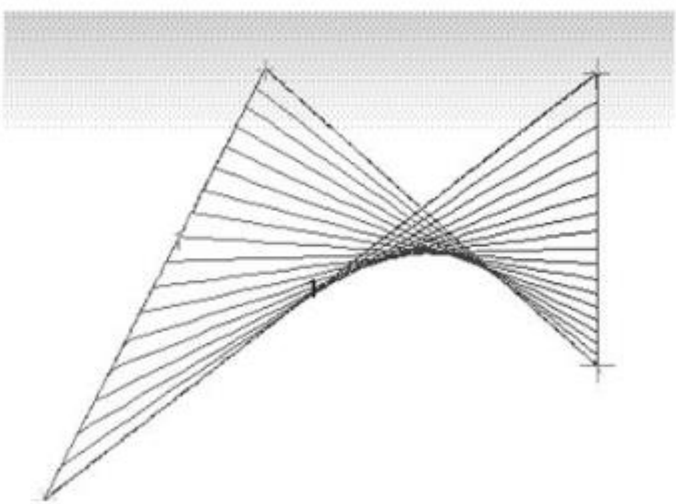


- **Bilinear $B\acute{e}zier$ Patch 를 이용하면 surface를 만들 수 있다**
- **a=[0, 1, 0], b=[1, 1, 0], c=[0, 0, 0], d=[1, 0, 0]**
- **식을 구해 보자**

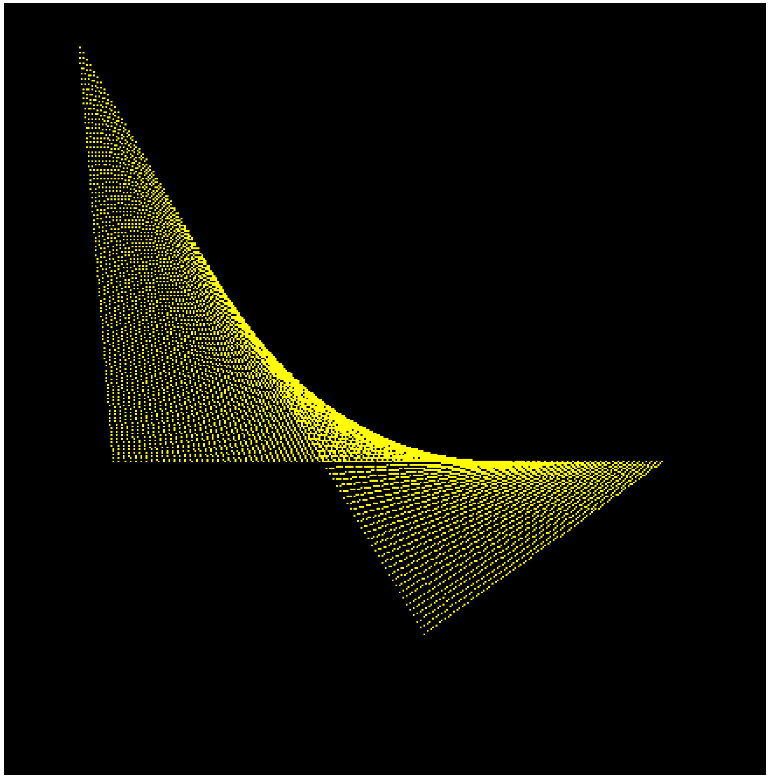
```
#include <GL/glut.h>          //  we will use GLUT (GL UTILITY TOOLKIT)
void Display(){
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1.0, 1.0, 0.0);
 glLoadIdentity();
gluLookAt (3.0,3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

 int i;
 int j;
 int NUM=30;
 double u;
 double v;
glBegin(GL_POINTS);
 for (i=0; i< NUM; i++)
 {

        for (j=0; j< NUM; j++)
        {
        u=double(i)/(NUM*1.0);
    v=double(j)/(NUM*1.0);
        glVertex3f( u, 1-v, 0);
        }
 }
 glEnd();

 glFlush();
 }

void resize(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, (float)w/(float)h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
}


int main(){
 glutInitWindowSize(600,600);
 glutInitWindowPosition(300,300);
 glutCreateWindow("OpenGL Hello World!");
 glutDisplayFunc(Display);
 glClearColor(0.0, 0.0, 0.0, 0.0);
  glutReshapeFunc(resize);
 glutMainLoop();
 return 0;
 }
```



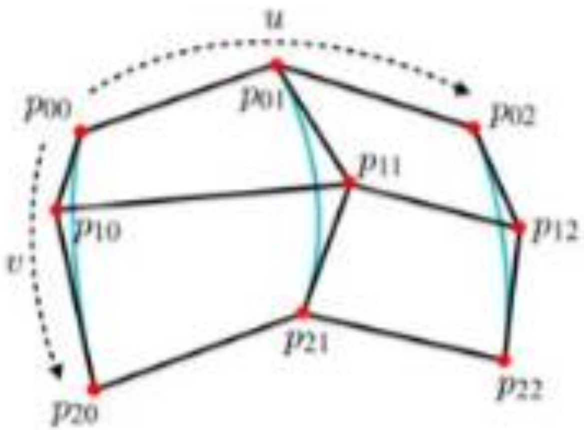
- Control points 위치에 따라서 다양한 surface를 만들 수도 있다
- 4개의 control point를 이용하여 Bilinear $B\acute{e}zier$ Patch 만듬
- a=[-1, 0, 0,], b=[1, 0, -2], c=[-1, 2, 0], d=[1, 0, 0]



```
#include <GL/glut.h>          //  we will use GLUT (GL UTILITY TOOLKIT)
void Display(){
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 0.0);
  glLoadIdentity();
gluLookAt (3.0,3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

 int i;
 int j;
 int NUM=100;
 double u;
 double v;
glBegin(GL_POINTS);
 for (i=0; i< NUM; i++)
 {

                        for (j=0; j< NUM; j++)
                        {
                        u=double(i)/(NUM*1.0);
    v=double(j)/(NUM*1.0);
                        glVertex3f( (1-u)*(1-v)*-1 + u*(1-v)*1 + v*(1-u)*-1 + u*v*1,  (1-u)*(1-v)*0 + u*(1-v)*0 + v*(1-u)*2 + u*v*0, -2*u*(1-v));
                        }
 }
  glEnd();

  glFlush();
 }

void resize(int w, int h)
{
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, (float)w/(float)h, 1.0, 20.0);
   glMatrixMode(GL_MODELVIEW);
}


int main(){
  glutInitWindowSize(400,400);
  glutInitWindowPosition(300,300);
  glutCreateWindow("OpenGL Hello World!");
  glutDisplayFunc(Display);
  glClearColor(0.0, 0.0, 0.0, 0.0);
    glutReshapeFunc(resize);
  glutMainLoop();
  return 0;
 }
```



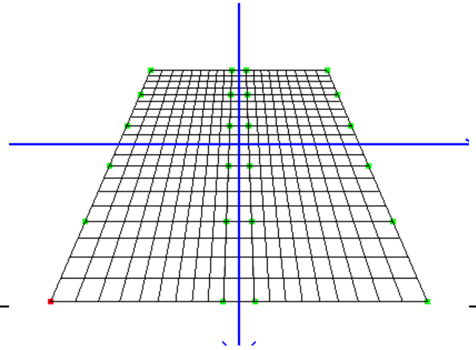
- Biquadratic $\mathbf{B\acute{e}zier}$ Patch (surface)
- 비슷한 방식으로 아래 그림과 같이 두 개의 파라미터 u, v를 사용하여 2차 $\mathbf{B\acute{e}zier}$ Patch 를 만들 수 있다
- 직접 유도해 보자
- 비슷한 방식으로 Bicubic $\mathbf{B\acute{e}zier}$ Patch 도 확장 가능하다



$$\begin{aligned} p(u,v) = {} & (1-u)^2(1-v)^2 p_{00} + 2u(1-u)(1-v)^2 p_{01} + u^2(1-v)^2 p_{02} + \\ & 2(1-u)^2 v(1-v) p_{10} + 4uv(1-u)(1-v) p_{11} + 2u^2 v(1-v) p_{12} + \\ & (1-u)^2 v^2 p_{20} + 2u(1-u)v^2 p_{21} + u^2 v^2 p_{22} \end{aligned}$$

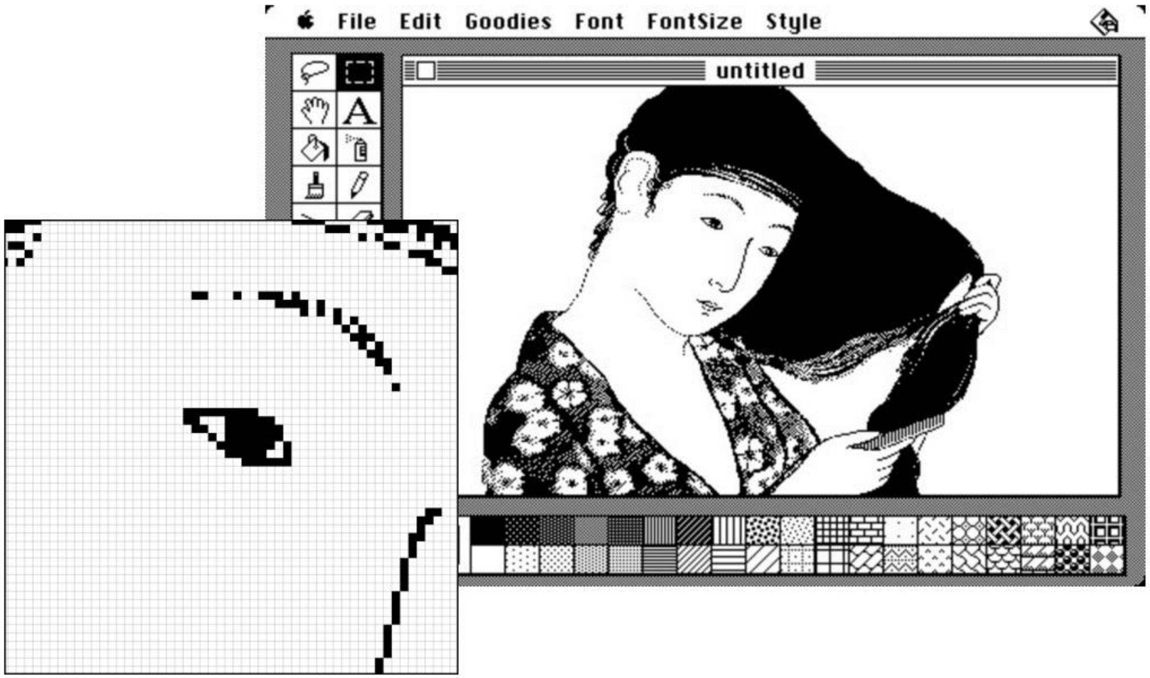
인천대학교

- *Bézier* surface OpenGL 코드 예
- https://www.dropbox.com/s/4e4cz5xfx3zzu36/bezier_surface.txt?dl=0
- space 키와 tab키로 control point 선택
- 오른쪽/왼쪽 키로 control point x축에서 위아래 이동
- 위/아래 키로 control point y축에서 위아래 이동
- page up/down 키로 control point z축에서 위아래 이동
- x,y,z키로 회전



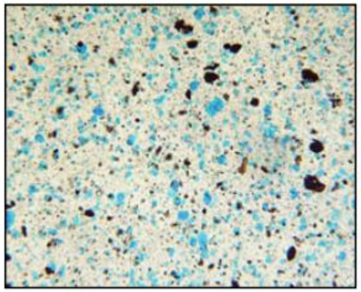
- **Modeling shapes with polygonal meshes**
- **(출처: 미국 CMU강의, CMU 15-462/662)**

- **To encode (bitmap) images, we used a regular grid of pixels**
- **If we zoom in far enough, blocks of colors**



- **But images are not fundamentally made of little squares**

Goyō Hashiguchi, *Kamisuki* (ca 1920)



photomicrograph of paint

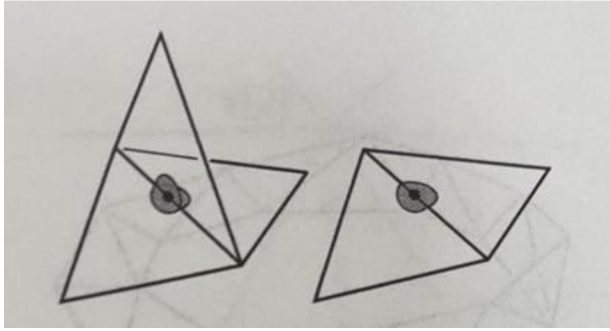
- **So why did we choose a square grid? Rather than dozens of possible alternatives? Triangles, squares, polygons**

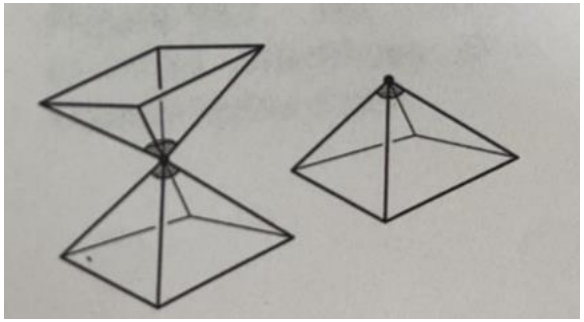
- **One reason: simplicity/efficiency**
  - E.g., always have four neighbors
  - Easy to index, easy to filter
- **Another reason: generality**
  - Can encode basically any image
- **Will see a similar story with geometry**

	(i,j-1)	
(i-1,j)	(i,j)	(i+1,j)
	(i,j+1)	

- Intuitively, **a surface is the boundary or “shell” of an object**
- **Think about the candy shell, not the chocolate**
- **Surfaces are manifold (매니폴드):**
    - If you zoom in far enough, can draw a regular coordinate grid (두 방향)
    - 확대해보면, 2D grid 처럼 보임 (x, y축)

- 교재에서의 manifold 설명
- Roughly speaking, a **manifold** is a surface in which a small neighborhood around any point could be smoothed out into a bit of **flat surface**





(왼쪽) non-manifold edge

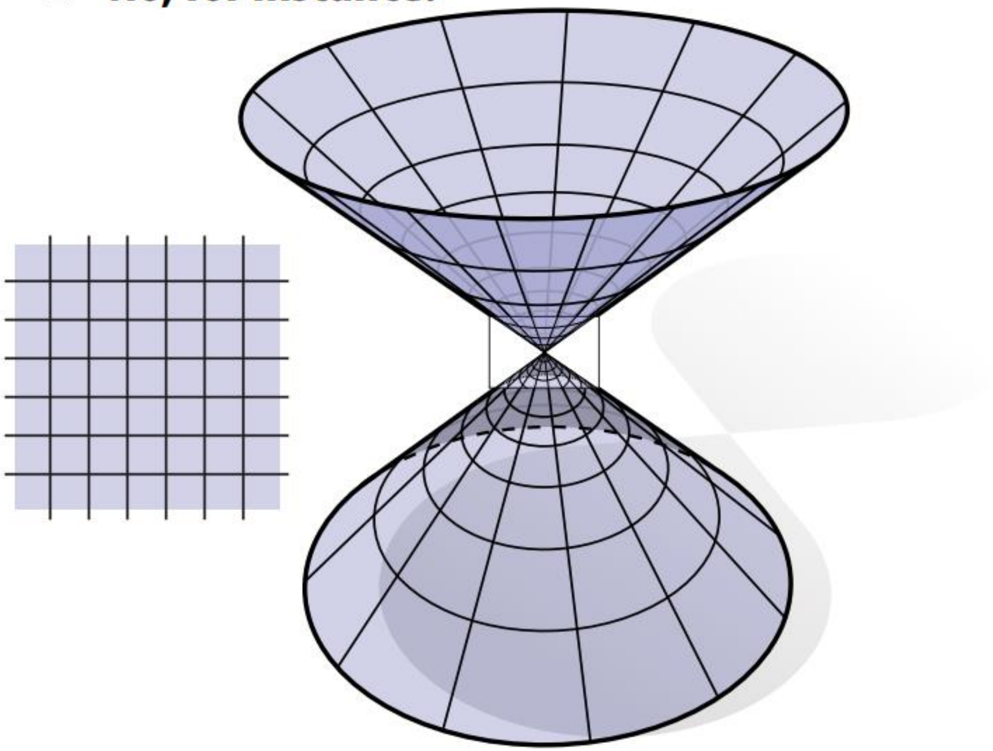
(오른쪽) manifold edge

(왼쪽) non-manifold vertex

(오른쪽) manifold vertex

- **Isn’t every shape manifold?**

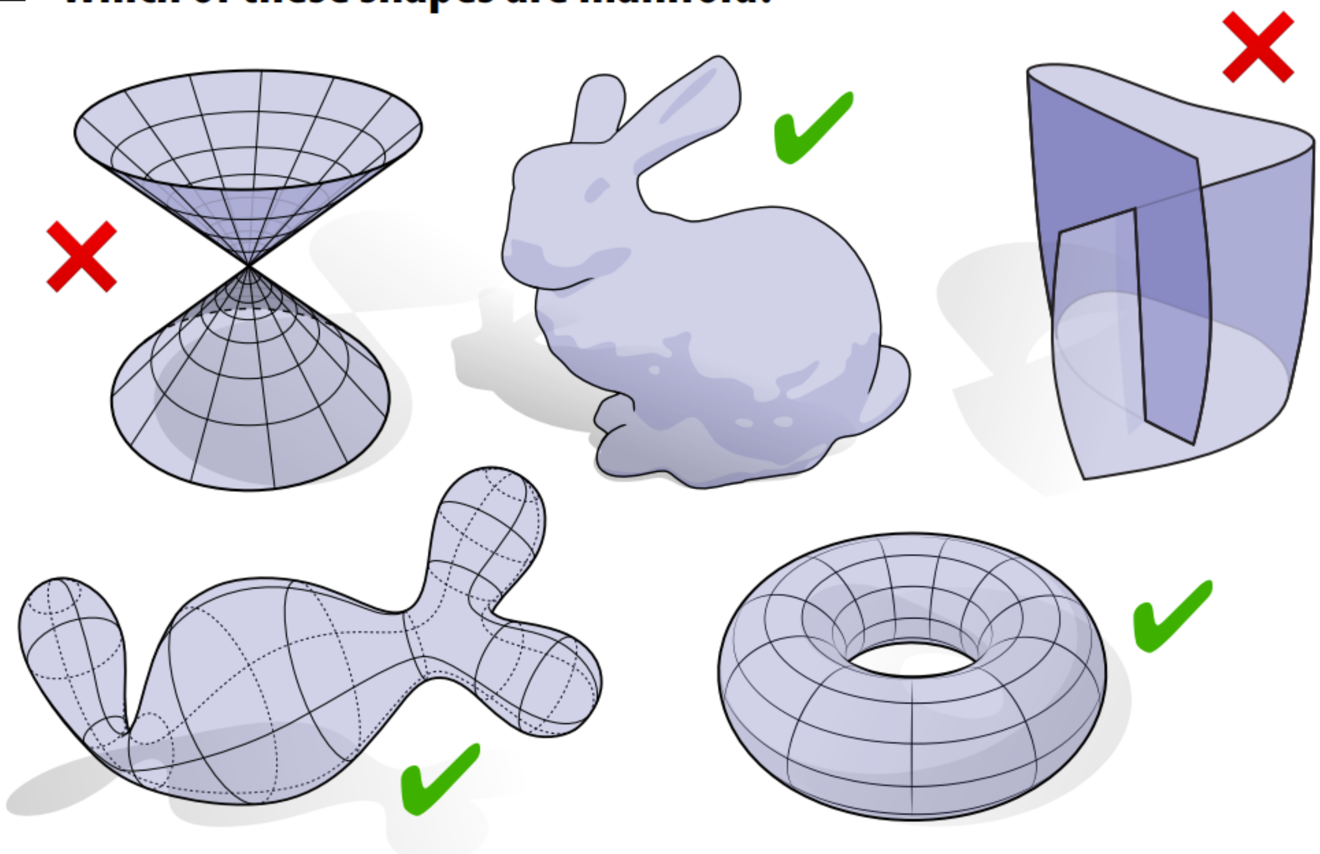
  - **No, for instance:**



Can’t draw ordinary 2D grid at center, no matter how close we get.

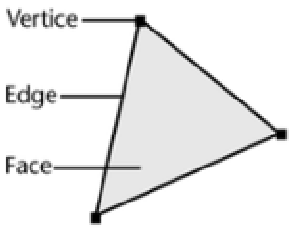
인천대학교

- Which of these shapes are manifold?



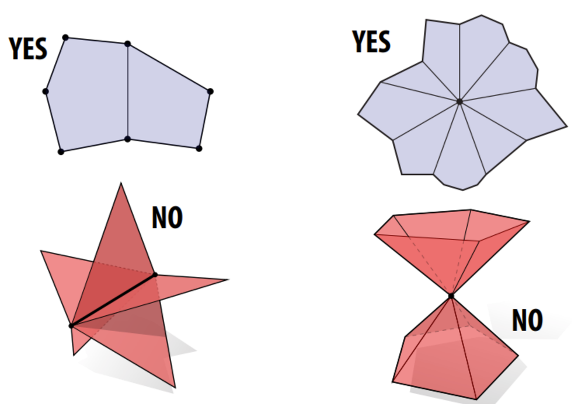
- **Polygonal meshes**

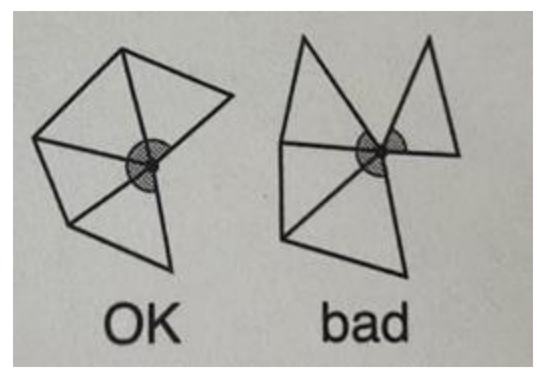
- Polygon: polygons are straight-sided shapes (3 or more sides), defined by vertices and the straight lines that connect them (edges)



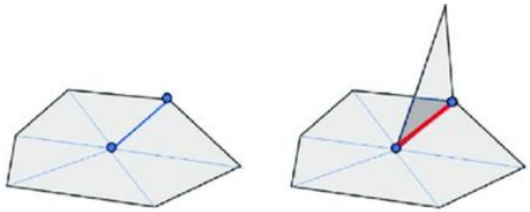
- Face: polygon의 내부 영역
- Polygon의 기본 요소: vertices, edges, face
- Polygon은 가장 단순하면서 더 많이 사용되는 2D 물체임
- 우리가 사용하는 polygon은 simple and planar polygon임

- A polygonal meshes (meshes) is a union of convex polygons satisfying the manifold condition
- 우리가 다루는 polyon mesh는 manifold 조건을 만족하는 mesh
- 이를 manifold mesh라고 부름
- Two easy conditions to check
- 1. Every (interior) edge is contained in only two polygons
- 2. The polygons containing each vertex make a single “fan”

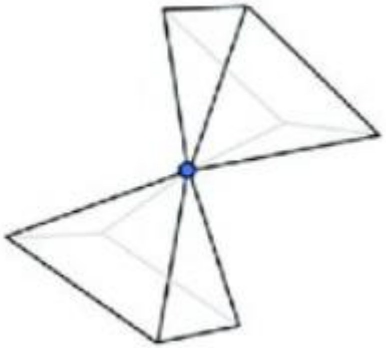




- (왼쪽) manifold edge (오른쪽) mon-manifold edge



- Non-manifold vertex



- **Why is the manifold assumption useful?**

Same motivation as for images

- make some assumptions about our geometry to keep data structures/algorithms simple and efficient
- many common cases, doesn't fundamentally limit what we can do with geometry

	(i,j-1)	
(i-1,j)	(i,j)	(i+1,j)
	(i,j+1)	

- Many graphics algorithms assume that meshes are manifold (manifold mesh 가정)
- 3D 프린터에서도 manifold mesh 사용
- 많은 Mesh visualization 및 editing SW에서는 non-manifold edge나 vertex를 찾아줌 (제거도)
- [MeshLab Basics: Non manifold elements - YouTube]

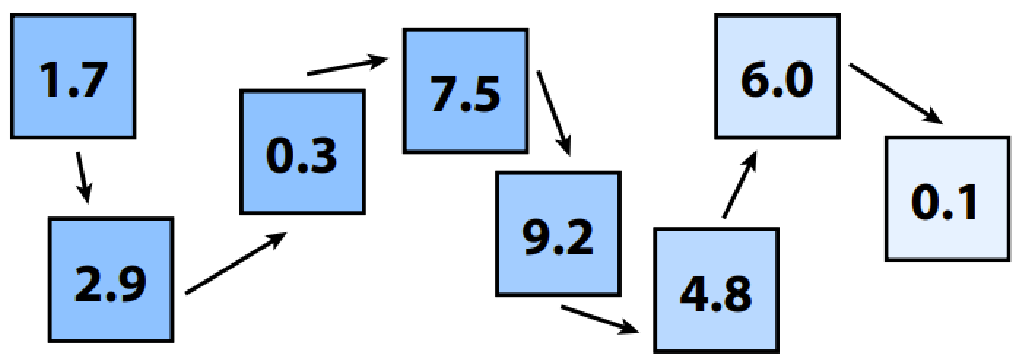
- **How do we actually encode all this data?**

Warm up: storing numbers

- Q: What data structures can we use to store a list of numbers?
- One idea: use an *array* (constant time lookup, coherent access)

1.7	2.9	0.3	7.5	9.2	4.8	6.0	0.1

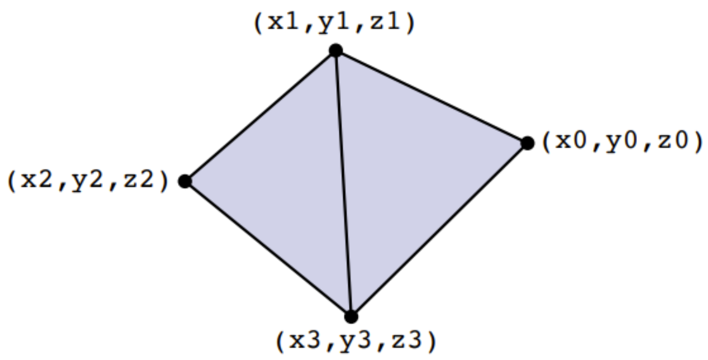
- Alternative: use a linked list (linear lookup, incoherent access)



- Q: Why bother with the linked list?
- A: For one, we can easily insert numbers wherever we like...

인천대학교

- **Encoding polygon meshes**
- **Most basic idea**
  - For each triangle, just store three coordinate
  - No other information about connectivity
- **Not much different from point cloud! Triangle cloud**
- **장점: really stupidly simple**
- **단점: redundant storage**
- **Hard to find neighbors**



x0,y0,z0	x1,y1,z1	x3,y3,z3
x1,y1,z1	x2,y2,z2	x3,y3,z3

인천대학교

Adjacency List (Array-like)

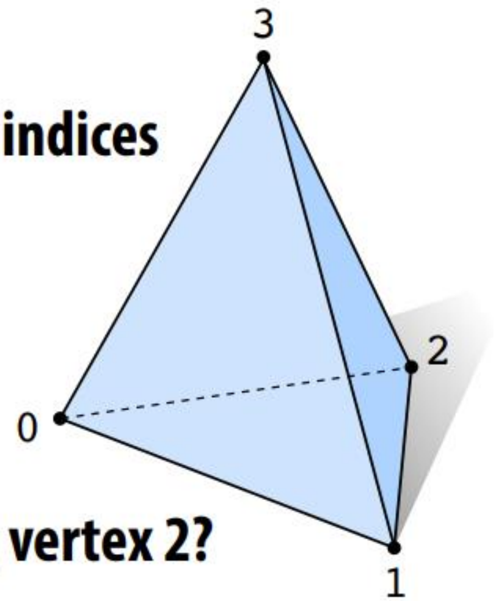
- Store triples of coordinates (x,y,z), tuples of indices
- E.g., tetrahedron:

VERTICES

	x	y	z
0:	−1	−1	−1
1:	1	−1	1
2:	1	1	−1
3:	−1	1	1

POLYGONS

i	j	k
0	2	1
0	3	2
3	0	1
3	1	2



- Q: How do we find all the polygons touching vertex 2?
- Ok, now consider a more complicated mesh:

~1 *billion* polygons

Mesh를 2개의 list로 나눔.

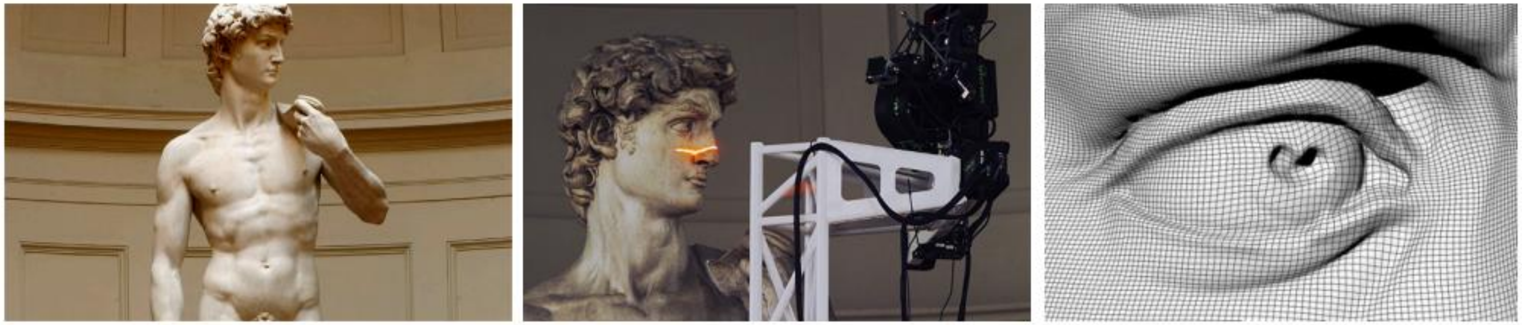
Vertex list와 connectivity (triangle) list

- Q: How do we find all the polygons touching vertex 2?
- **Algorithm**

1. Go down the list of polygons
2. One polygon at a time check whether it contains vertex 2
3. Move to the next polygon

- **Ok, now consider a more complicated mesh:**

~1 *billion* polygons



Very **expensive to find the neighboring polygons! (What's the cost?)**

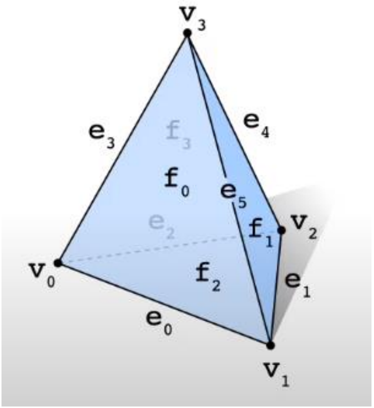
- Incidence matrix
- **If we want to know who our neighbors are, why not just store a list of neighbors**
- **Can encode all neighbor information via incidence matrices**
- **E.g., tetrahedron (vertex list는 앞과 동일)**

VERTEX↔EDGE

	v0	v1	v2	v3
e0	1	1	0	0
e1	0	1	1	0
e2	1	0	1	0
e3	1	0	0	1
e4	0	0	1	1
e5	0	1	0	1

EDGE↔FACE

	e0	e1	e2	e3	e4	e5
f0	1	0	0	1	0	1
f1	0	1	0	0	1	1
f2	1	1	1	0	0	0
f3	0	0	1	1	1	0



- **1 means “touches”; 0 means “does not touch”**
- **Still large storage cost, but**

인천대학교 University of Incheon

- 예: face f1를 이루는 3 vertex를 어떻게 찾는가?
- A: 1. edge-face matrix의 f1에서 1인 부분을 찾음

=> e1, e4, e5

2. vertex-edge matrix에서 e1, e4, e5가 1인 부분을 찾음

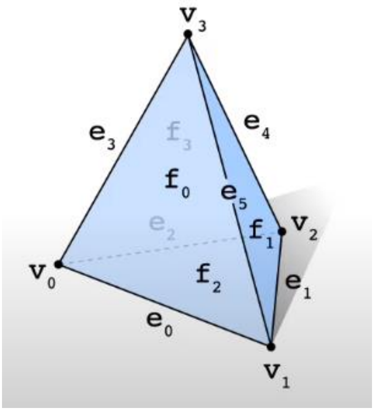
=> {v1, v2, v3}

VERTEX↔EDGE

	v0	v1	v2	v3
e0	1	1	0	0
e1	0	1	1	0
e2	1	0	1	0
e3	1	0	0	1
e4	0	0	1	1
e5	0	1	0	1

EDGE↔FACE

	e0	e1	e2	e3	e4	e5
f0	1	0	0	1	0	1
f1	0	1	0	0	1	1
f2	1	1	1	0	0	0
f3	0	0	1	1	1	0



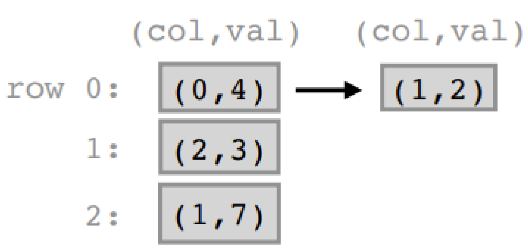
인천대학교

- **Sparse matrix data structures**
- **How do we actually sotre a “sparse matrix”?**
- **Lots of possible data structures**
- **Associative array form (row, column) to value**

$$\begin{array}{c|c} & \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} & \begin{bmatrix} 4 & 2 & 0 \\ 0 & 0 & 3 \\ 0 & 7 & 0 \end{bmatrix} \end{array}$$

```
(row,col)   val
(0,0) -> 4
(0,1) -> 2
(1,2) -> 3
(2,1) -> 7
```

- **Array of linked list (one per row)**



인천대학교

- **Importing meshes**

- **3D mesh File format**
- **확장자 .3ds, .ply, .stl, .obj, .vtk,….**
- **예: obj file format**
- **3D mesh를 표현하는**
- **https://en.wikipedia.org/wiki/Wavefront_.obj_file**
- **http://www.andrewnoske.com/wiki/OBJ_file_format**
- **https://dirsig.cis.rit.edu/docs/new/obj.html**

- **예: obj (.obj) is a simple data-format that represents 3D geometry alone – namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normal, and the faces that make each polygon defined as a list of vertices, and texture vertices**

- obj mesh 파일 포맷의 예
- # comment line (무시됨)
- # vertex 위치 정보 (x, y, z)
- v x y z
- …
- # vertex의 texture 정보 (u, v). 0에서 1사이 값으로. (s, t)와 유사
- vt u v
- …
- # vertex의 법선 벡터 정보
- vn x y z
- …
- # face의 정보 (vertex 들의 index 정보 줌)
- f v1 v2 v3
- …

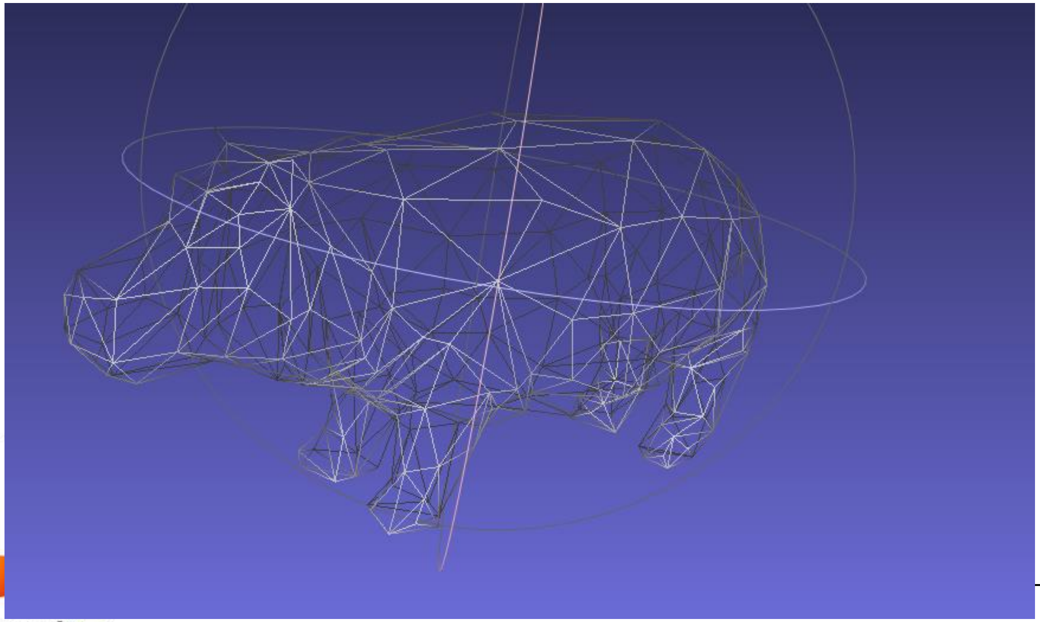
- https://www.dropbox.com/s/40yq3jzu71f8uy8/cube.obj?dl=0
- https://www.dropbox.com/s/acbya4b9t6nf22z/Laurana50k.ply?dl=0
- https://www.meshlab.net/#download
- Stanford graphics lab
- http://graphics.stanford.edu/data/3Dscanrep/

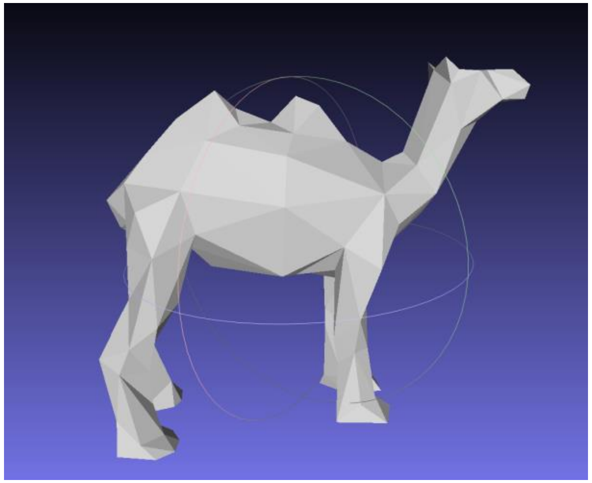
■ 예: cube.obj : vn (vertex normal), smooth shading

```
# Object cube.obj
# Vertices: 8
# Faces: 6
####
v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
v  0.0  1.0  1.0
v  1.0  0.0  0.0
v  1.0  0.0  1.0
v  1.0  1.0  0.0
v  1.0  1.0  1.0

vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
vn  0.0  1.0  0.0
vn  0.0 -1.0  0.0
vn  1.0  0.0  0.0
vn -1.0  0.0  0.0

f 1 2 4 3
f 2 6 8 4
f 6 5 7 8
f 5 1 3 7
f 3 4 8 7
f 5 6 2 1
# 6 faces, 0 coords texture

# End of File
```

- MeshLab: 3D triangular mesh를 processing하는 오픈소스 시스템
- 여러가지 mesh 관련된 작업, editing, visualization 가능
- 샘플 mesh 제공
- https://www.meshlab.net/

- **교재에서 제공하는 mesh file들**
- **Chapter10/OBJModels/Meshes**
- **camel.obj, 340개의 polygon**
- **Hippo.obj, 498개의 polygon**





- 오른쪽 menu에 wireframe 모드
- point cloud