Chap 2. Graphics Programming

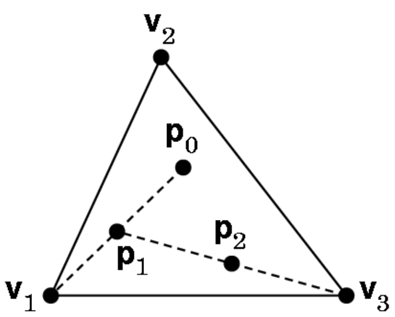
동의대학교
멀티미디어공학과

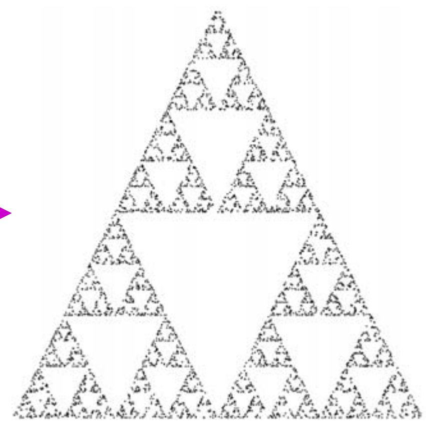
Hyoungseok B. Kim

2.1 The Sierpinski Gasket

Sierpinski gasket

- 예제 문제로 사용
  - 원래, Fractal geometry 에서 정의
- 만드는 방법
  - 삼각형을 그린다.
  - 삼각형 내부에 random하게 점 $\mathbf{P}_i$ 를 선택, 출력
  - random 하게 꼭지점 중의 하나 $\mathbf{V}_i$ 선택
  - $\mathbf{V}_i$ 와 $\mathbf{P}_i$ 의 중점을 $\mathbf{P}_{i+1}$ 로 선택, 출력
  - 위 과정을 반복





Program Outline

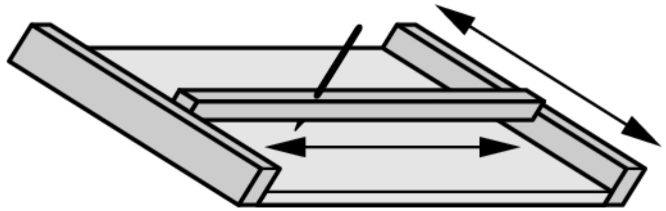
- 전체 구조
  - void main(void) {

```
void main(void) {
    initialize_the_system( );
    for (some_number_of_points) {
        pt = generate_a_point( );              // 좌표 계산
        display_the_point( pt );               // 출력
    }
    cleanup( );
}
```

- 제일 먼저 할 일 ?
  - 점을 어떻게 출력할 것인가?

Pen-Plotter Model

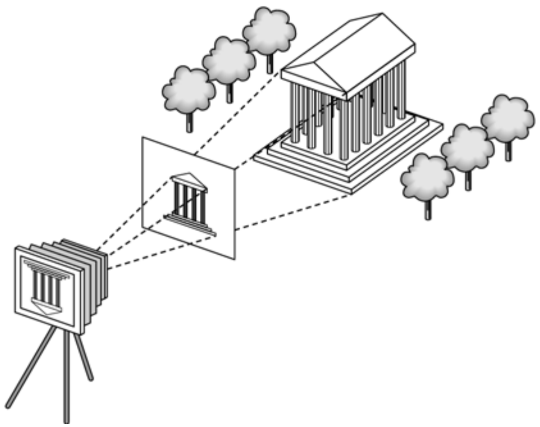
- pen-plotter

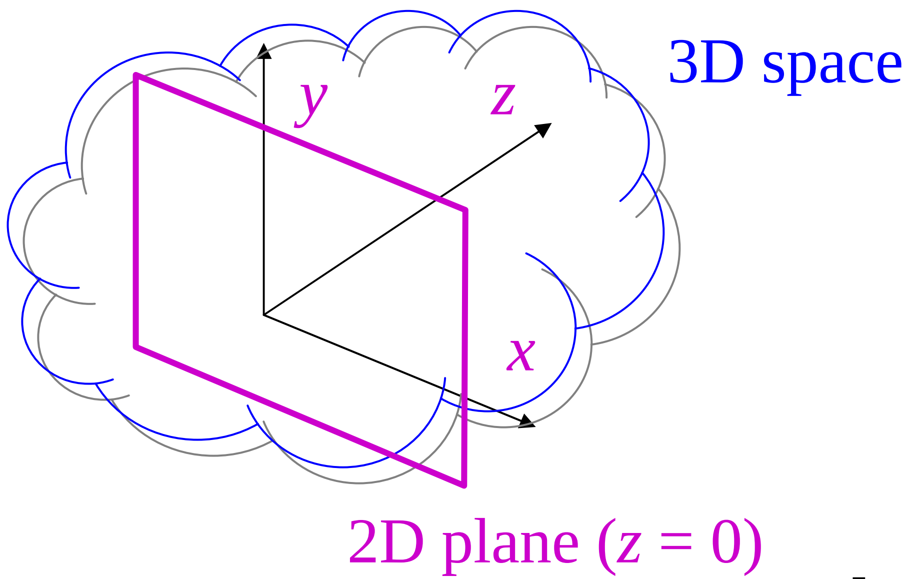


  - 2D 종이 위에 펜을 움직여서 출력
  - moveto(*x*, *y*); 정해진 위치로 펜을 이동
  - lineto(*x*, *y*); 정해진 위치까지 선분 출력
- 가장 오래된 graphics output model
  - 장점 : 간단. 2D 종이, 2D 화면에 적합한 model
    - printer 용 언어, 초기 graphics system 에서 사용
    - PostScript, PDF, LOGO, GKS, …
  - 단점 : 3D model 에는 부적합

2D in a 3D Model

- 3D system에서의 2D 처리
  - 2D 는 3D의 특별한 경우이다
  - 3D 좌표 : (x, y, z)
  - 2D로 해석할 때는 $z = 0$: $(x, y, 0)$
  - 간단하게 : (x, y)

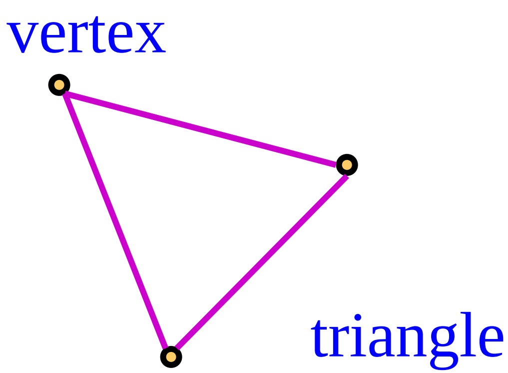




Vertex

- space 상의 위치 1개
  - graphics 에서는 2D, 3D, 4D space 사용
- 표기법 : column vector

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



- geometric objects
  - point : vertex 1개로 정의
  - line segment : vertex 2개로 정의
  - triangle : vertex 3개로 정의
- point 와 헷갈리지 말 것

Vertex 정의 in OpenGL

- OpenGL 에서 vertex 를 정의하는 함수

  glVertex[*n*][*t*][v](…);

  – *n* : number of coordinates
    - *n* = 2, 3, 4

  – *t* : coordinate type
    - *t* = i (integer), f (float), d (double), …

  – v : vector or not
    - v 가 붙으면, vector (= array) form

- 모든 OpenGL 함수는 gl 로 시작

OpenGL suffixes

suffix	data type	C-language	OpenGL type
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int / long	GLint, GLsizei
f	32-bit floating pt.	float	GLfloat, GLclampf
d	64-bit floating pt.	double	GLdouble, GLclampd
ub	8-bit unsigned int	unsigned char	GLubyte, GLboolean
us	16-bit unsigned int	unsigned shrot	GLushort
ui	32-bit unsigned int	unsigned int / unsigned long	GLuint, GLenum, GLbitfield

Examples

- void glVertex2i(GLint xi, GLint yi);
  - 사용 예:　　　glVertex2i(2, 3);

- void glVertex3f(GLfloat x, GLfloat y, GLfloat y);
  - 사용 예:　　　glVertex3f(1.5, 2.3, 3.0);

- void glVertex3dv(GLdouble v[3]);
  - 사용 예:　　　GLdouble vertex[3] = { 1, 2, 3 };
    glVertex3dv(vertex);

Object 정의 in OpenGL

- geometric object 의 정의
  - vertex가 여러 개 모여서 하나의 object
  - glBegin(*TYPE*);

```
glBegin(TYPE);
  glVertex*(…);
  glVertex*(…);
  …        /* 다른 함수도 가능 */
glEnd( );
```

- 사용 예
  - glBegin(GL_LINES);

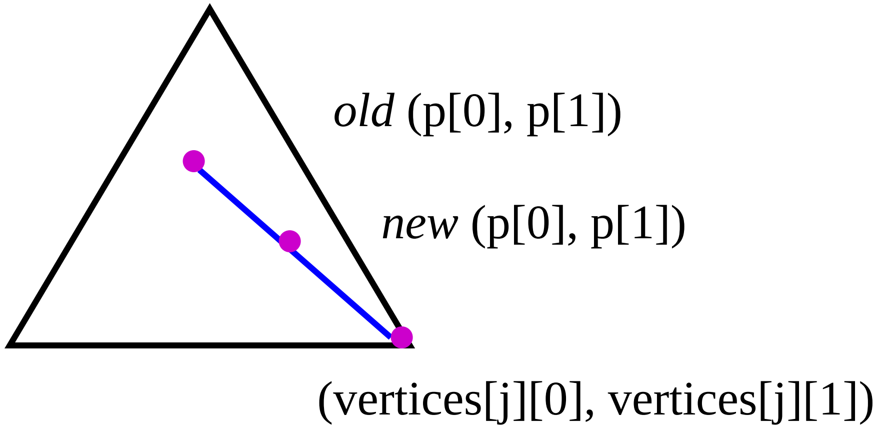
```
glBegin(GL_LINES);
  glVertex2f(x1, y1);
  glVertex2f(x2, y2);
glEnd( );
```

Object-oriented Paradigm

- graphics program들은 object-oriented paradigm에 적합
- Java3D : fully object-oriented library
  - Point3 old(2, 1, 3);
    Vector3 vec(1, 0, 0);
    Point3 new = old + vec;
- OpenGL : not object-oriented !
  - C-based library
  - 대안은 array 뿐
  - typedef GLfloat Point2[2];
    Point2 p = { 2, 3 };
    …
    glVertex2fv(p);

Sierpinski gasket 구현 예제

```
void display( void ) {
   point2 vertices[3]={{0.0,0.0},{250.0,500.0},{500.0,0.0}}; /* A triangle */
   point2 p ={75.0,50.0};            /* An arbitrary initial point inside triangle */
   int i, j, k;

   for (k=0; k<5000; k++) {
      j=rand( ) % 3;     /* pick a vertex at random */
      /* Compute point halfway between selected vertex and old point */
      p[0] = (p[0] + vertices[j][0]) / 2.0;
      p[1] = (p[1] + vertices[j][1]) / 2.0;
      /* plot new point */
      glBegin(GL_POINTS);
         glVertex2fv(p);
      glEnd( );
   }
   glFlush( ); /* flush buffers */
}
```



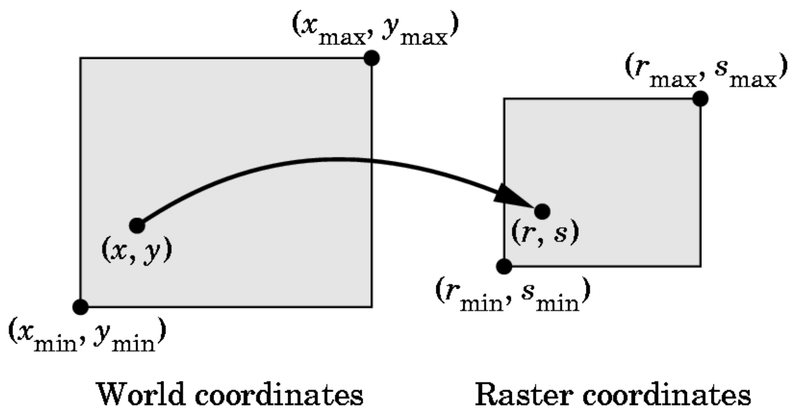
Some questions …

- int rand(void); integer random number generator
  - see <stdlib.h>, void srand(void);
- void glFlush(void); flush the graphics pipeline

- we still have some questions…
  - in what colors are we drawing ?
  - where on the screen does our image appear ?
  - how large will the image be ?
  - how do we create the window for our image ?
  - how much of our infinite 2D plane will appear ?
  - how long will the image remain on the screen ?

Coordinate Systems

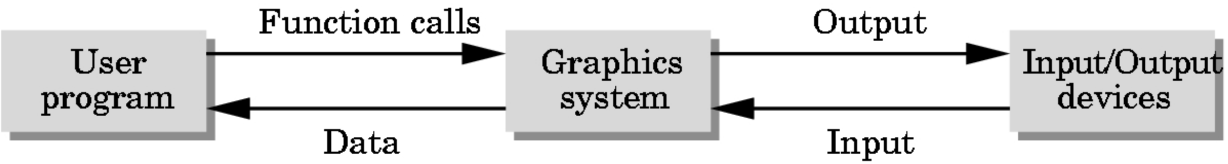
- 2D 좌표 사용 : 좌표를 어떻게 해석할 것인가?
- device-independent coordinate system
- world coordinate system = problem coordinate system
  - 그림을 정의하는 좌표계(OpenGL 좌표계)
- device coordinate system
  - = physical-device / raster / screen coordinate system
  - 그림이 그려지는 좌표계



2.2 The OpenGL API

Graphics API

- OpenGL 도 이러한 특성을 가짐



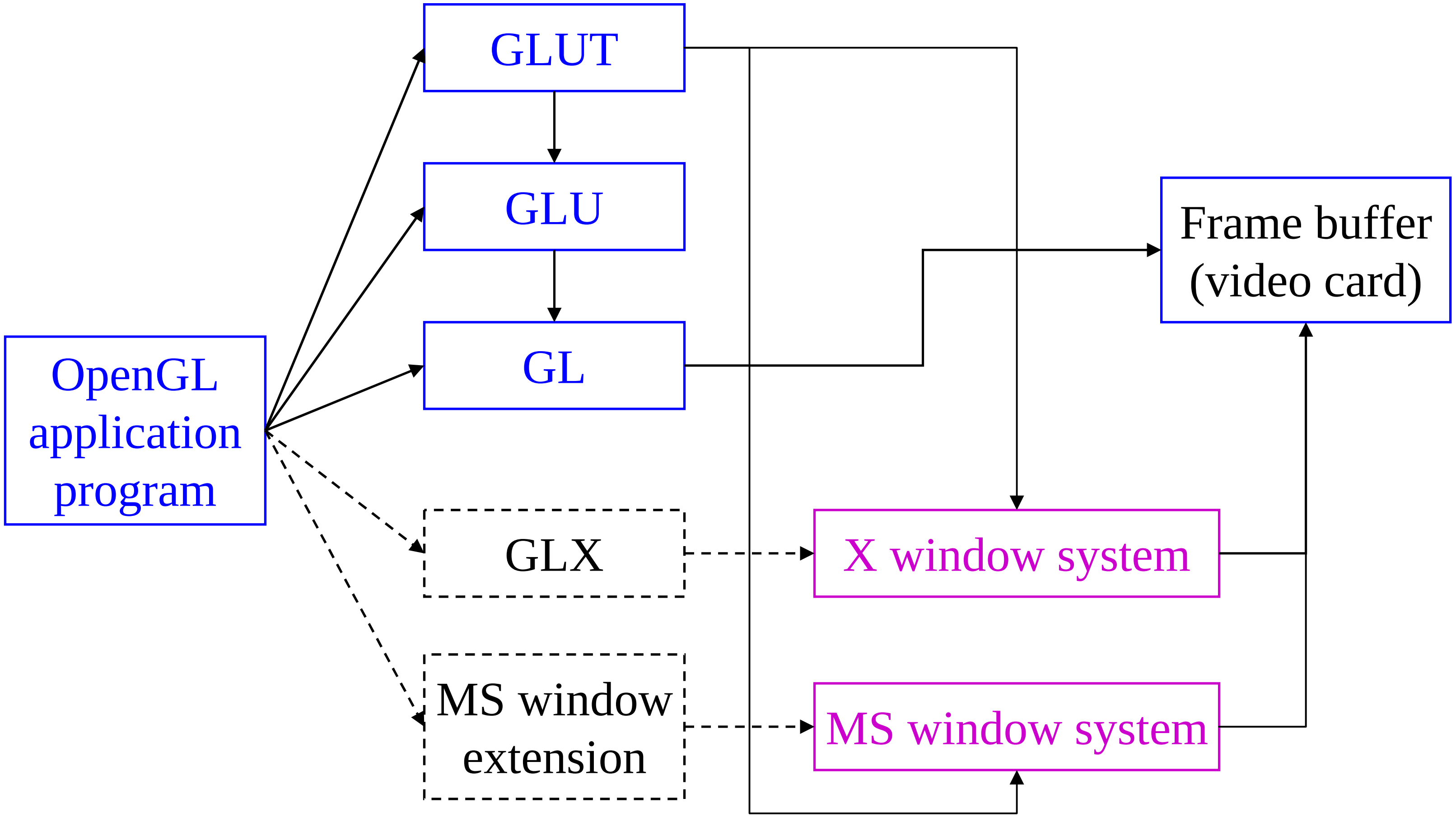
- Graphics API = hundreds of functions + α
- function 의 구분
  - primitive functions 무엇을 출력 ?
  - attribute functions 어떤 모양으로 출력 ?
  - viewing functions 카메라 설정
  - transformation functions 물체 위치 변환
  - input functions 사용자 입력
  - control functions window 관리

OpenGL 구성

- 특성
  - C-based library (not object-oriented)

- 3개의 library 로 구성
  - GL : graphics library
    - H/W에서 지원하여야 하는 기본 primitives
  - GLU : graphics utility library
    - S/W로 지원해도 되는 확장 primitives
  - GLUT : GL utility toolkit
    - window system들을 위한 지원 함수들

OpenGL 구성

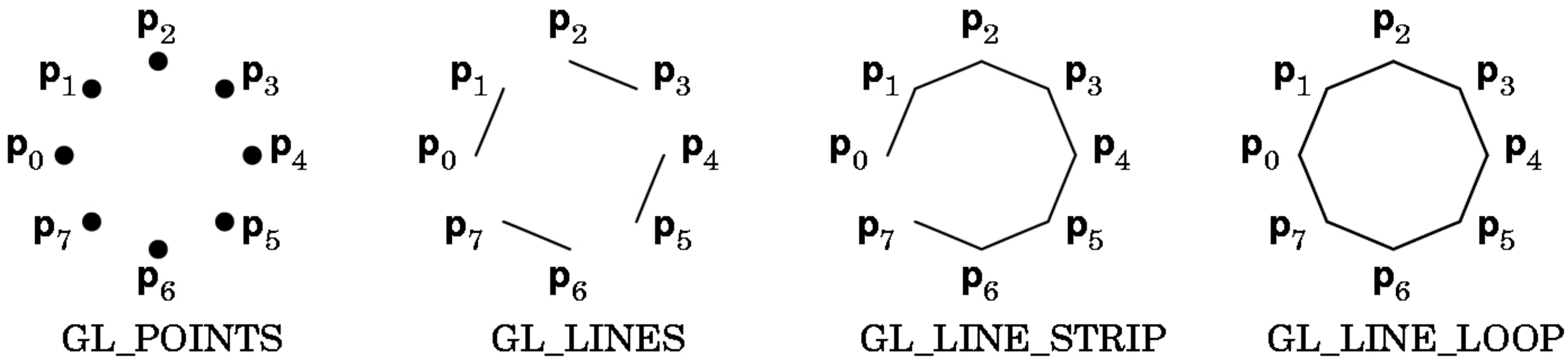


GLUT
GLU
GL
Frame buffer
(video card)
OpenGL
application
program
GLX
X window system
MS window
extension
MS window system

2.3 Primitives and Attributes

Line Primitives

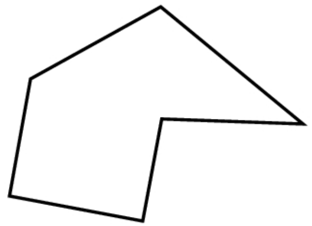
- 기본 구조
  - glBegin(type);
    glVertex*(…);
    …
    glVertex*(…);
    glEnd( );
- type 별 출력 예제



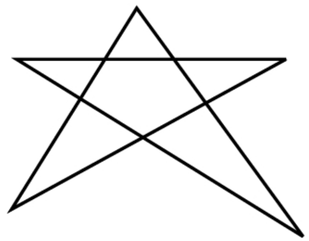
Polygon

- polygon : an object with the border
  - polygon = loop + interior
- assumption : simple, convex, and flat
  - simple : edge 끼리의 intersection 없음
  - convex : 볼록 다각형
  - flat : 3D 에서 하나의 평면 상에 위치해야
    - flatness 보장을 위해, triangle을 주로 사용

simple

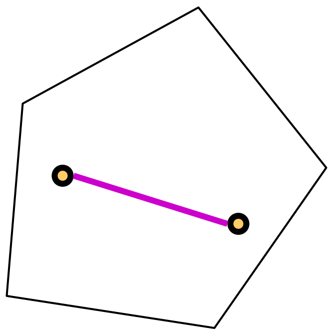


(a)

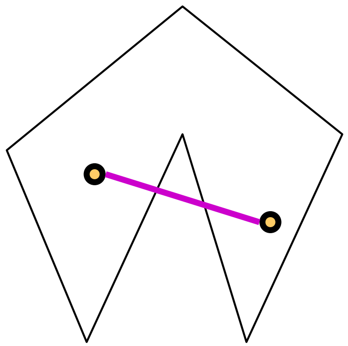


(b)

nonsimple



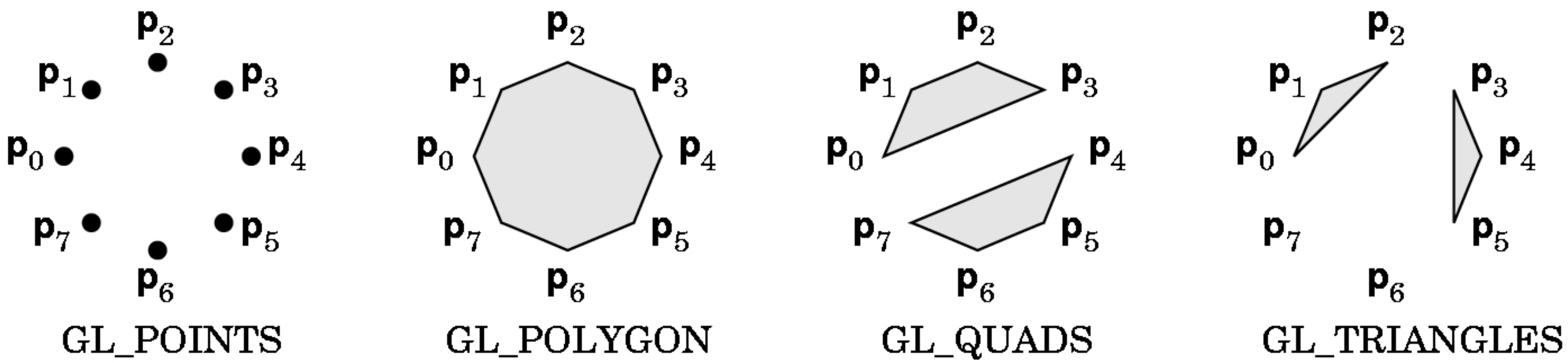
convex



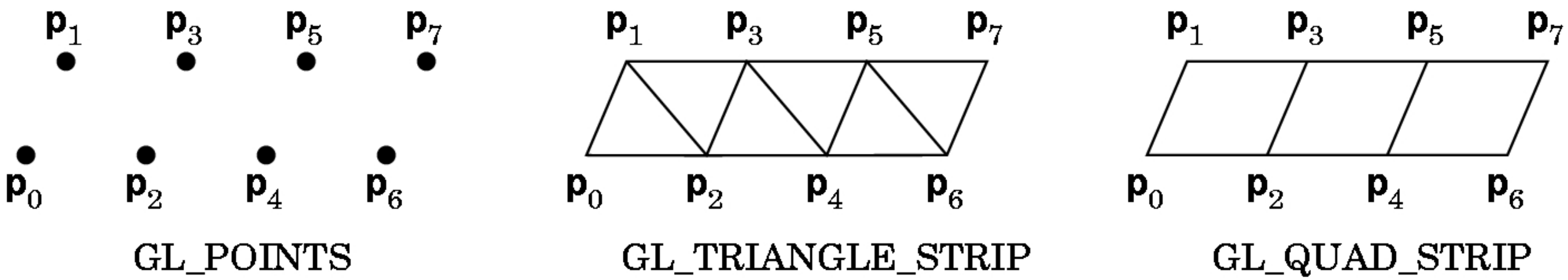
concave

Polygon Primitives

- 일반적인 경우



- strips : 속도를 높이기 위해



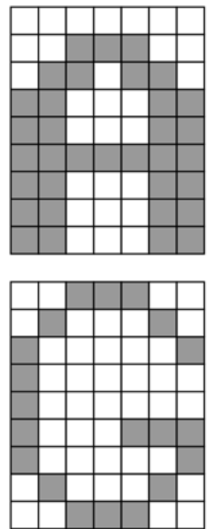
Text

- text in computer graphics is problematic.
  - 3D text 는 일반적인 text 보다 훨씬 복잡
  - OpenGL : no text primitive (use window primitives)
  - GLUT : minimal support
    - glutBitmapCharacter(…);
- stroke font (= outline font) : graphics 에서 주로 사용
  - character = boundary를 정의하는 수학 함수들
  - 확대/축소에 편리
  - 제작/출력에 많은 시간 필요

Computer

Graphics

- raster font (= bitmap font) : text-based application 용
  - character = raster pattern



Curved Objects

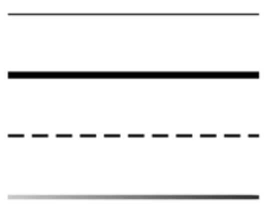
- curved object 의 처리 방법

- tessellation
  - polyline / polygon 으로 근사(approximation)

- 수학적으로 직접 표현
  - chap. 10 에서 설명

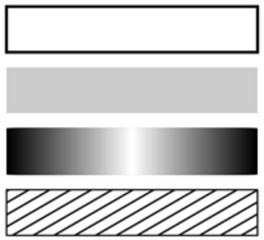
Attributes

- attribute = any property that determines
  how a geometric primitve is to be rendered.
- point : color, size
- line : color, thickness, type (solid, dashed, dotted)
- polygon : fill color, fill pattern
- text : height, width, font, style (bold, italic)



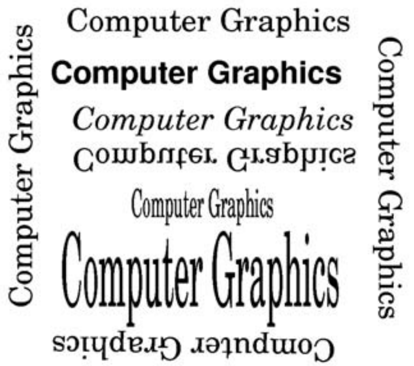
(a)

line attributes



(b)

polygon attributes

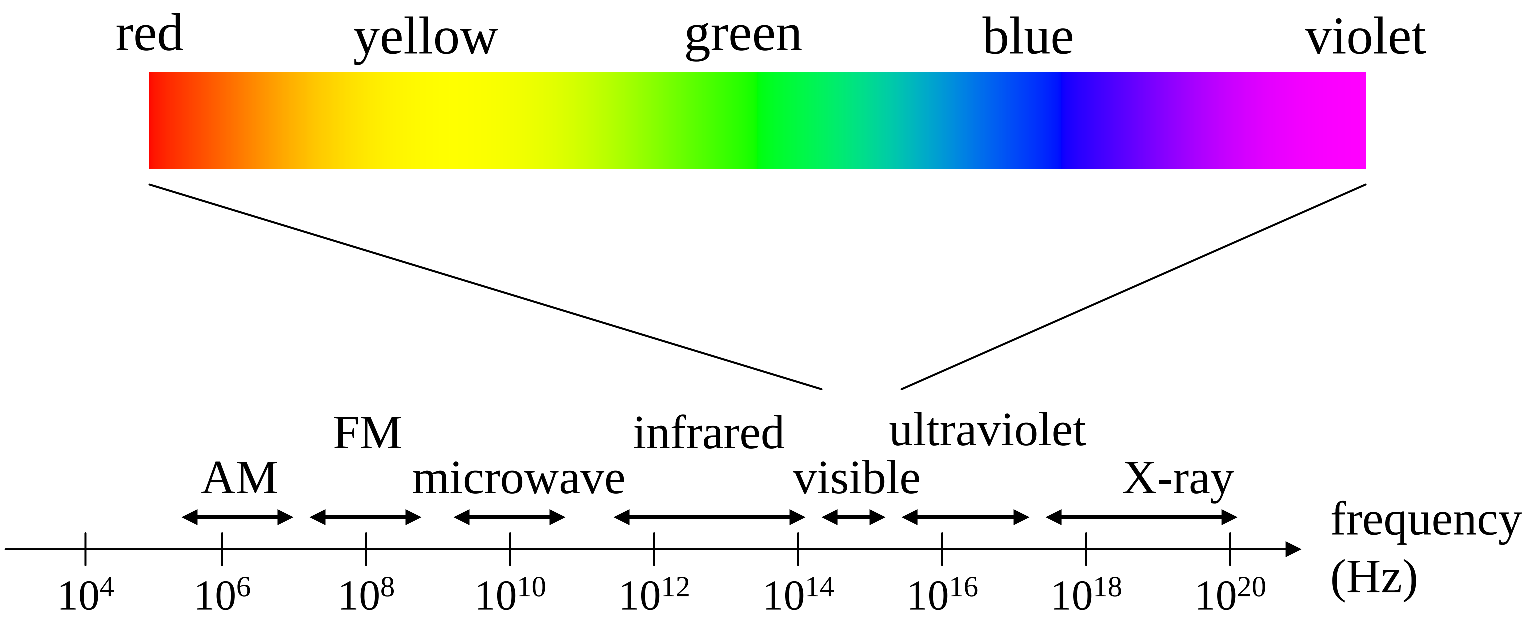


text attributes

2.4 Color

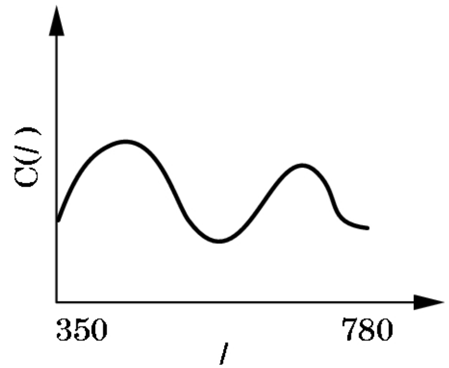
Light

- electromagnetic wave
  – 흔히 가시광선(visible light)를 의미

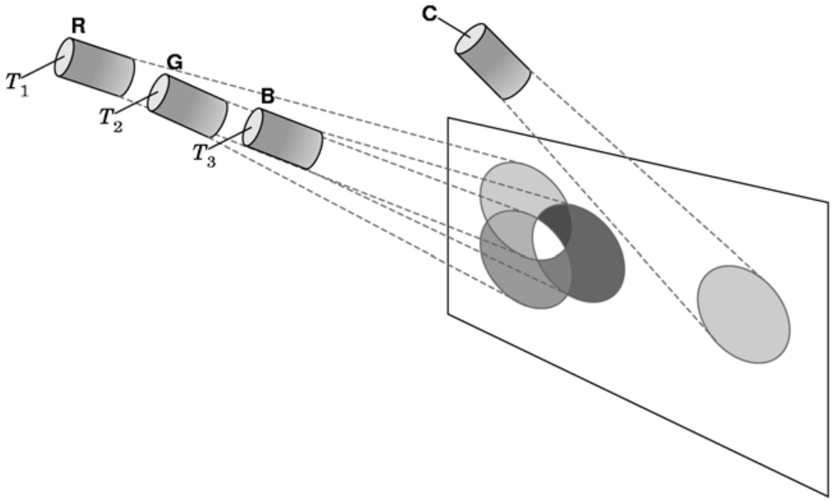


Color

- C(λ) : wave length λ 에 대한 energy distribution
  - C(λ) 에 따라, color 결정



- additive color model
  - C(λ) 끼리 더할 수 있음
- three-color theory
  - $\mathbf{C} = T_1\,\mathbf{R} + T_2\,\mathbf{G} + T_3\,\mathbf{B}$

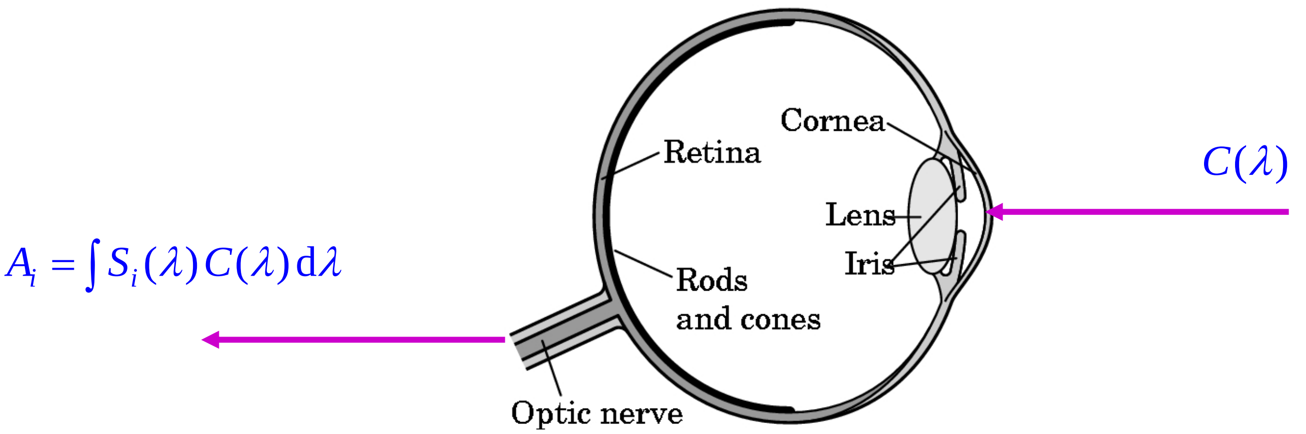


Human Visual System

- cones : red, green, blue 에 민감하게 반응
- sensitive curve $S_i(\lambda)$: wavelength에 따른 반응 정도
- brain perception values

$$A_i = \int S_i(\lambda)C(\lambda)\,\mathrm{d}\lambda \quad i = \text{red, green, blue}$$

- three-color theory의 기본 이론
  - $(A_{red}, A_{green}, A_{blue})$ 값들이 같으면, 같은 color로 인식



Color Model

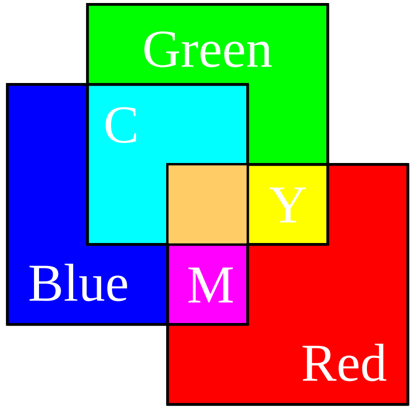
- color model
  - color를 computer H/W, S/W에서 표현하는 방법
  - 용도에 따라, 다양 :   RGB, CMY, YIQ, CIE, …

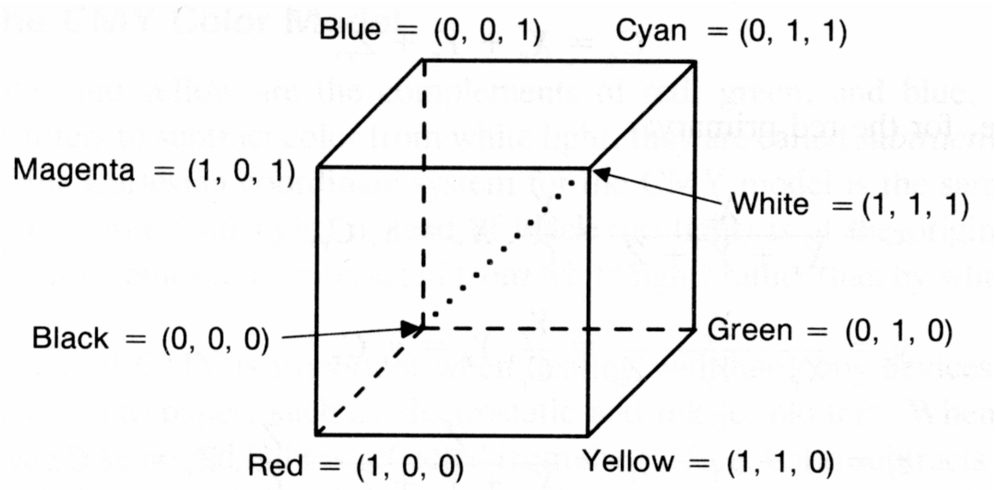
- color gamut
  - 특정 color model 이 생성 가능한 모든 color

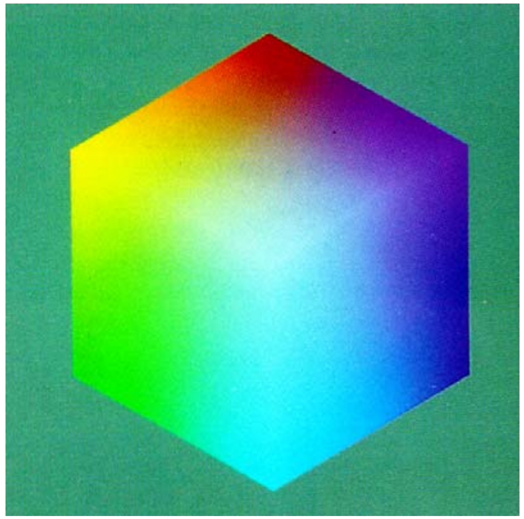
- color solid (= color cube)
  - color model 에서 흔히 three primary colors 사용
  - three primary color에 의한 3차원 cube
  - color gamut 표현 가능

RGB color model

- Red, Green, Blue
  - tri-stimulus theory
  - 눈에 가장 민감한 3가지 색상
- RGB cube : 각각이 0 ~ 1 까지



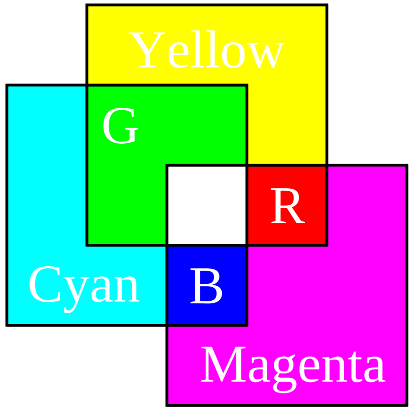




CMY, CMYK color model

- hard copy 기계에서는 잉크 사용
  - subtractive system 감산 색계
  - 흰 종이 위에
    cyan, magenta, yellow 사용

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}, \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$



- CMYK color model : K (black) 을 첨가
  - 이론상, cyan + magenta + yellow = black
  - 실제로는 dark gray
  - 해결책 :black(K) ink를 별도로

RGB vs. CMY

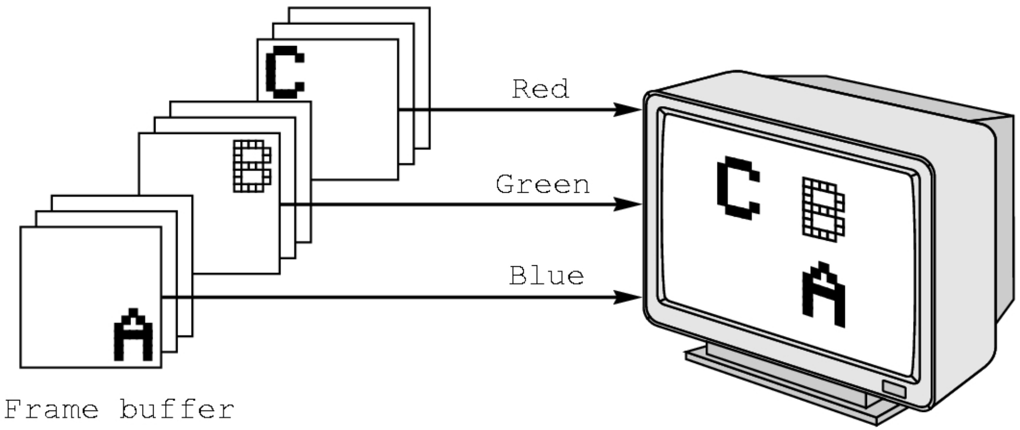
- RGB color system
- additive primaries
  - 더하면 밝아진다
- monitor 기준
  - 형광 물질로 R, G, B 사용

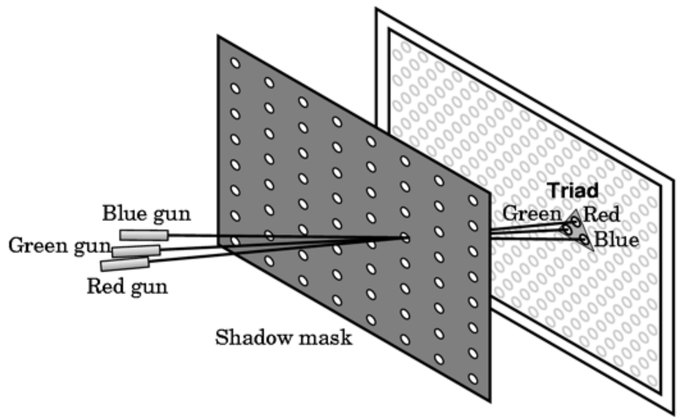
- CMY color system
- subtractive primiaries
  - 더하면 어두워진다
- printer 기준
  - ink로 C, M, Y 사용

- graphics는 주로 RGB 기준

Direct color system

- 기본적인 video card 구조
  - 3개의 전자총, 3개의 frame buffer
  - frame buffer 마다, pixel 당 *n* bit 할당
    - $2^n \times 2^n \times 2^n$ colors = 2^{3n} colors
  - 3*n* can be 8, 12, 24, …
  - 3*n* = 24 : true color system





Direct color system

- OpenGL functions
  - 3*n* 값은 system 마다 틀리다
  - color 설정은 RGB cube 기준
  - red, green, blue 모두 0.0 ~ 1.0

- void glColor*( );
- void glColor3f(GLclampf red, GLclampf green, GLclampf blue);
  - 현재 색상 정의
  - GLclampf : 0.0 보다 작으면 0.0, 1.0보다 크면 1.0
  - see OpenGL manual

Direct color system

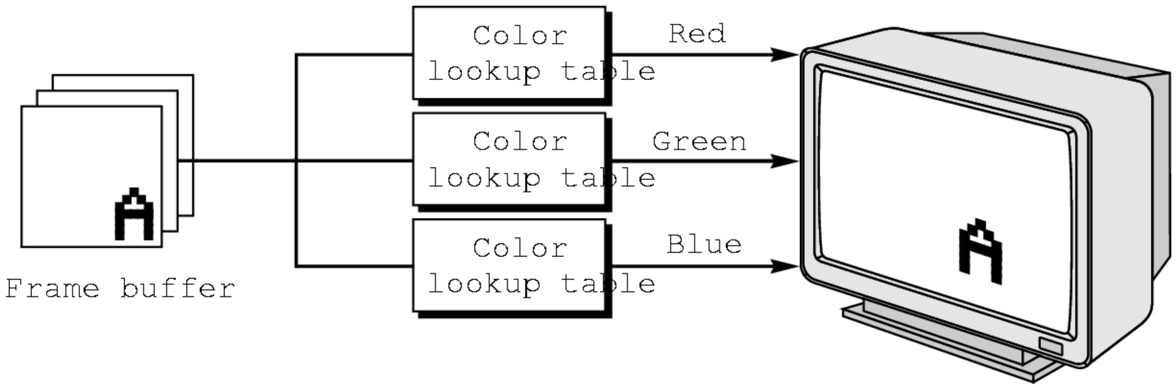
- RGBA color model
  - RGB + A (alpha channel)
  - alpha channel 은 opacity value : image 합성에 사용
  - A = 1.0 이 보통의 경우
- void glColor4f(red, green, blue, alpha);
- void glClearColor(GLclampf red, GLclampf green,
  GLclampf blue, GLclampf alpha);
  - clear color 설정 (= background color)
- void glClear(GLbitfield mask);
  - mask = GL_COLOR_BUFFER_BIT 이면,
    frame buffer 전체를 clear color로

Indexed color system

- frame buffer size 를 줄이는 방법
- color lookup table (LUT) (= palette)
  - 2^{3m} bit 로 color 값 2^k 개를 저장
- frame buffer : k bit index 값 저장
- 2^{3m} color 중에서 2^k 개만 동시 표현

Input		Red	Green	Blue
0		0	0	0
1		2^m 2 1	0	0
.		0	2^m 2 1	0
.		.	.	.
.		.	.	.
2^k 2 1		.	.	.
		m bits	m bits	m bits

color LUT



Indexed color system

- why indexed color system ?
  - image 표현에 필요한 frame buffer size 축소
  - image file format에서 사용 : GIF, BMP, …

- OpenGL functions
  - LUT의 setting은 window system / GLUT 가 담당

- void glIndex*(…);
  - current color를 LUT의 해당 index로 설정
- void glutSetColor(int index, GLfloat red, green, blue);
  - LUT의 해당 index를 새로운 color로

2.5 Viewing

Image Generation

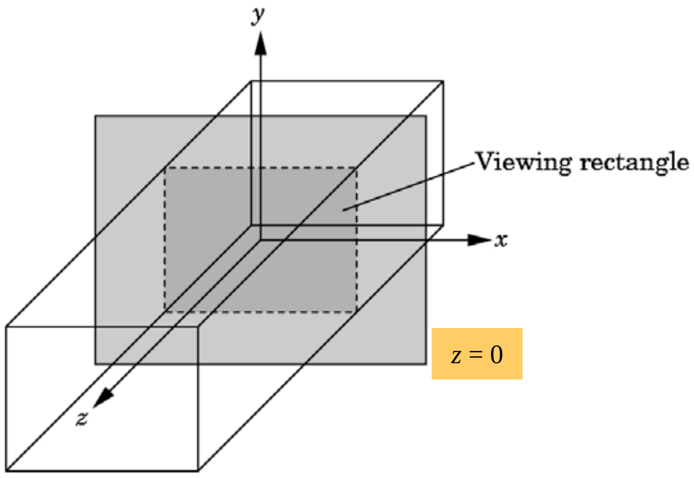
- camera로 object를 촬영

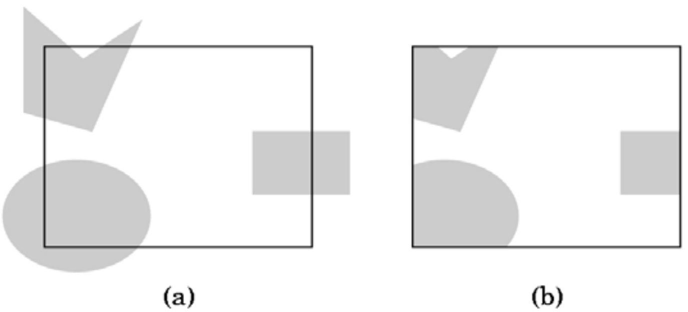
- graphics program에서는
  - object들을 배치
  - camera가 특정 위치에서 촬영

- viewing
  - graphics 에서,
    synthetic camera 의 위치 설정

2D Viewing

- 2D plane 상의 어느 부분이 보여야 하는가
- viewing rectangle
  - = clipping rectangle
  - 화면에 나올, 2D plane (z = 0) 상의 사각형 영역
- clipping
  - 화면에 나오지 않는 부분을 제거 (clipped out)



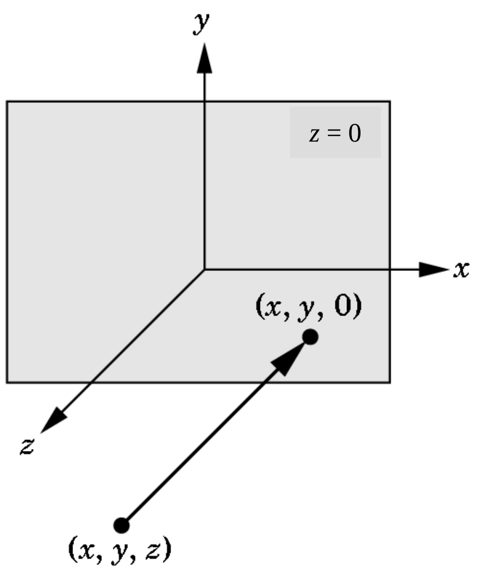


clipping operation

Orthographic View

- 3D viewing 의 일종
  - 2D viewing을 단순히 3D로 확장
  - 주의 : 모든 빛은 *z*축에 평행. 실제 camera와는 다름

$$(x, y, z) \rightarrow (x, y, 0)$$



  - OpenGL default :
    - $[-1, 1] \times [-1, 1] \times [-1, 1]$ 을 화면에 표시

Orthographic View

- OpenGL function
  - void glOrtho(GLdouble left, right,
    GLdouble bottom, top,
    GLdouble near, far);
  - void gluOrtho2D(GLdouble left, right,
    GLdouble bottom, top);
    - near = –1.0, far = 1.0 인 경우로 해석

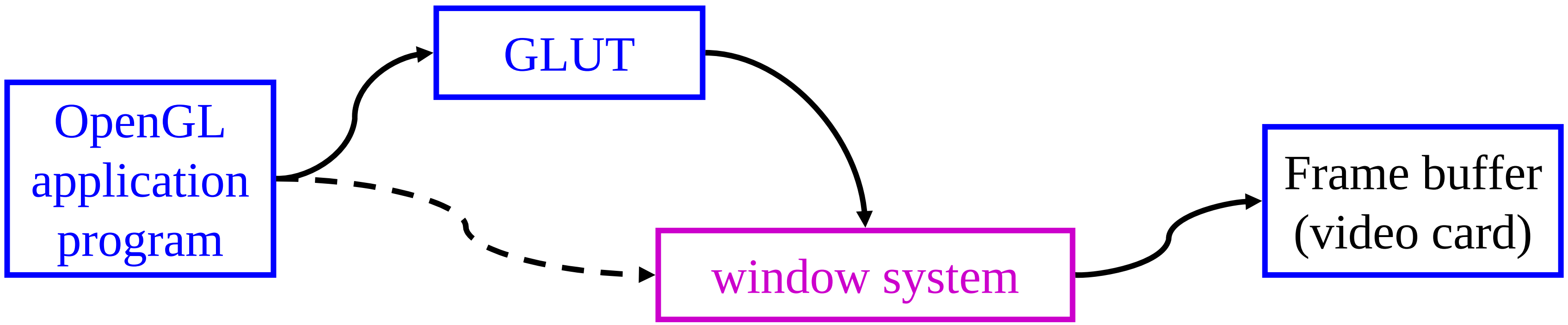
Matrix Handling

- camera, object 의 위치/방향 제어
  - = matrix handling 행렬 연산
- OpenGL has: two matrix mode
  - model-view matrices : object 용
  - projection matrices : camera 용
  - 각각의 역할이 다름 ⇒ chap 4.
- 간단한 예제 [0, 500] × [0,500] 인 경우
  - glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);
    glMatrixMode(GL_MODELVIEW);

2.6 Control Functions

Window System

- 현재, 다양한 window system 사용 중
  - 예 : X window, Macintosh, Microsoft window
  - OpenGL 관점에서는
    window 생성, 제어 방법이 완전히 다름
  - 해결책 : GLUT
    - 어디서나 작동하는 window control 방법 제공

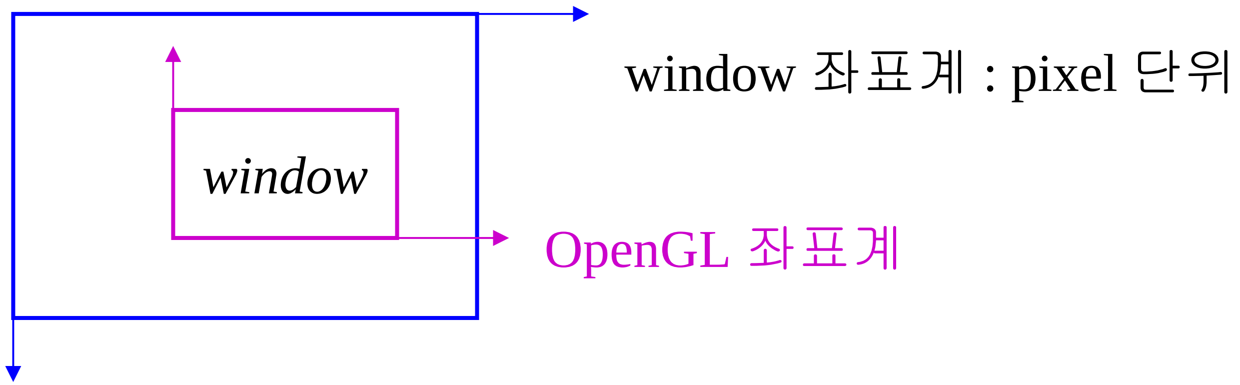


GLUT functions

- void glutInit(int* argcp, char* argv[]);
  - initialization. GLUT option 해석 가능
- int glutCreateWindow(char* title);
  - window 생성 (화면에는 표시되지 않음)
- void glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
  - display mode
  - RGB : direct color 사용
  - DEPTH : z-buffer 사용 (뒤에 설명)
- void glutInitWindowSize(int width, int height);
- void glutInitWindowPosition(int x, int y);
  - (x, y) 위치에서, width × height 크기로 window 생성

Coordinate system의 차이

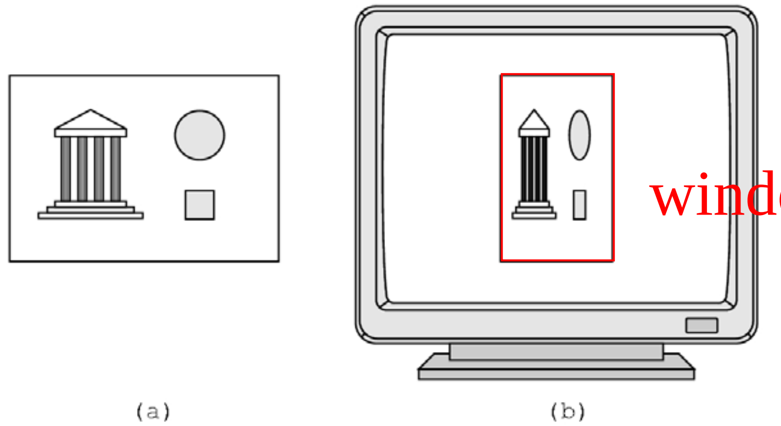
- 대부분의 window system
  - 원점이 upper left (뒤집어진 좌표계)
    - glutWindowPosition(…) : window system 기준
- 대부분의 고급 graphics library
  - OpenGL, PostScript, …
  - 기하학에서 쓰는 좌표계 그대로
    - glVertex3f(…) : OpenGL 기준



Viewport

- aspect ratio
  - window의 화면 비율
  - 단순 mapping 일 때는, mismatched image 가능

clipping rectangle

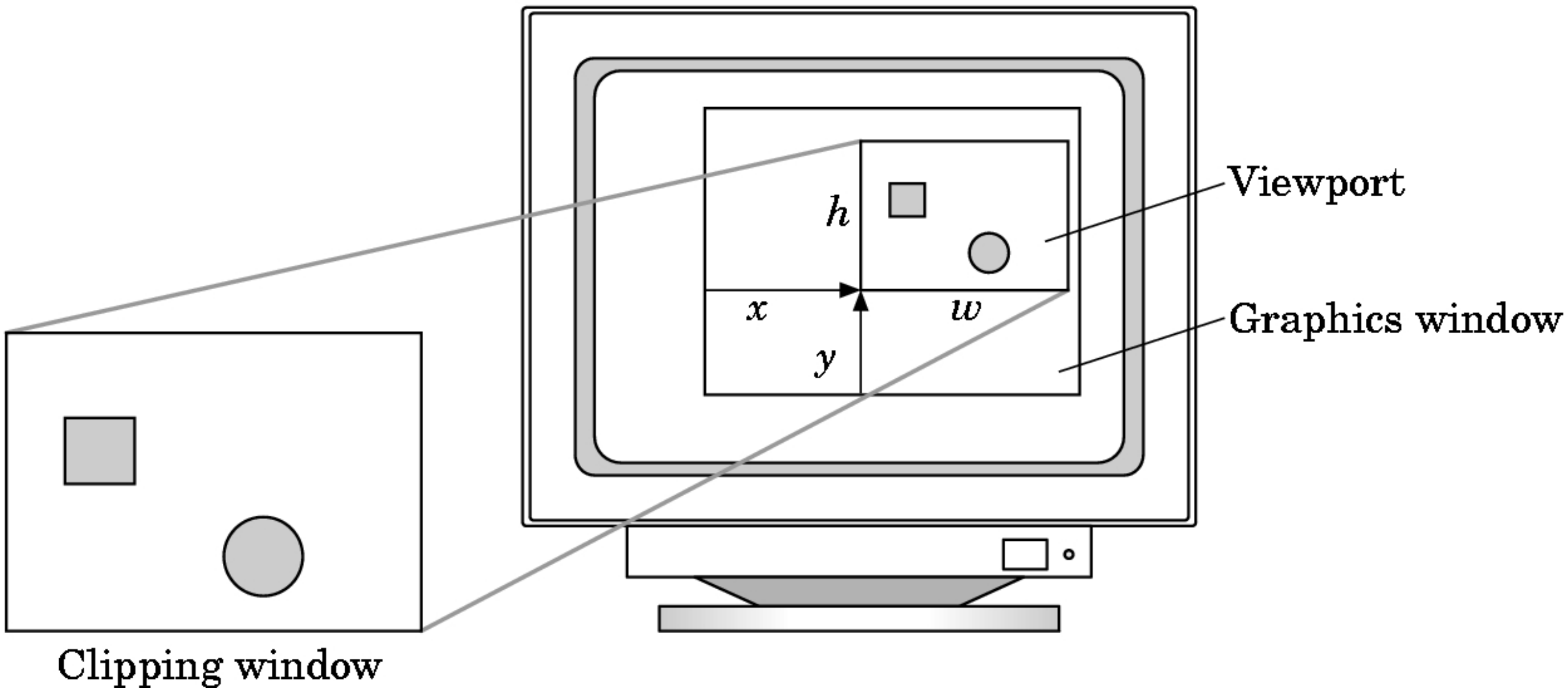


window

- viewport
  - window 영역 중에서,
    clipping rectangle 이 표시될 부분
- void glViewport(GLint x, y, GLsizei w, h);
  - (x, y) 위치에서, w × h 영역에 출력

Viewport

- 설정 예제



Graphics Program의 특징

- 일반적인 프로그램 : 출력이 끝나면, program 끝
- graphics 프로그램 : 출력을 그대로 유지해야
  - 해결책 : 화면을 그렸으면, infinite loop로

- 또다른 상황
  - window 가 가려졌다가, 다시 나타나면,
  - graphics window 전체를 새로 그려야 한다
  - 해결책 : display callback

- callback : 미리 등록해 두면,
  필요한 상황에서 자동으로 call 되는 function

GLUT 에서의 해결책

- void glutMainLoop(void);
  - 모든 설정이 끝난 시점에서, infinite loop 로 들어감
  - 보통, OpenGL program의 마지막 수행 함수

- void glutDisplayFunc(void (*func)(void));
  - 화면을 그려야 할 때 마다 호출되는 callback 함수 등록
  - callback 함수는 void *functionName*(void); 형태

Main function

```
#include <GL/glut.h>                          // GLUT 사용

void myInit(void) { … }        // 따로 작성 (sec 2.7)
void display(void) { … }       // 따로 작성 (sec 2.7)

void main(int argc, char* argv[]) {       // OpenGL program은 대부분 이런 구조
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Sierpinski Gasket");
  glutDisplayFunc(display);                 // callback 등록
  myInit( );                                // 미리 설정할 것들 처리
  glutMainLoop( );
}
```

2.7 The Gasket Program

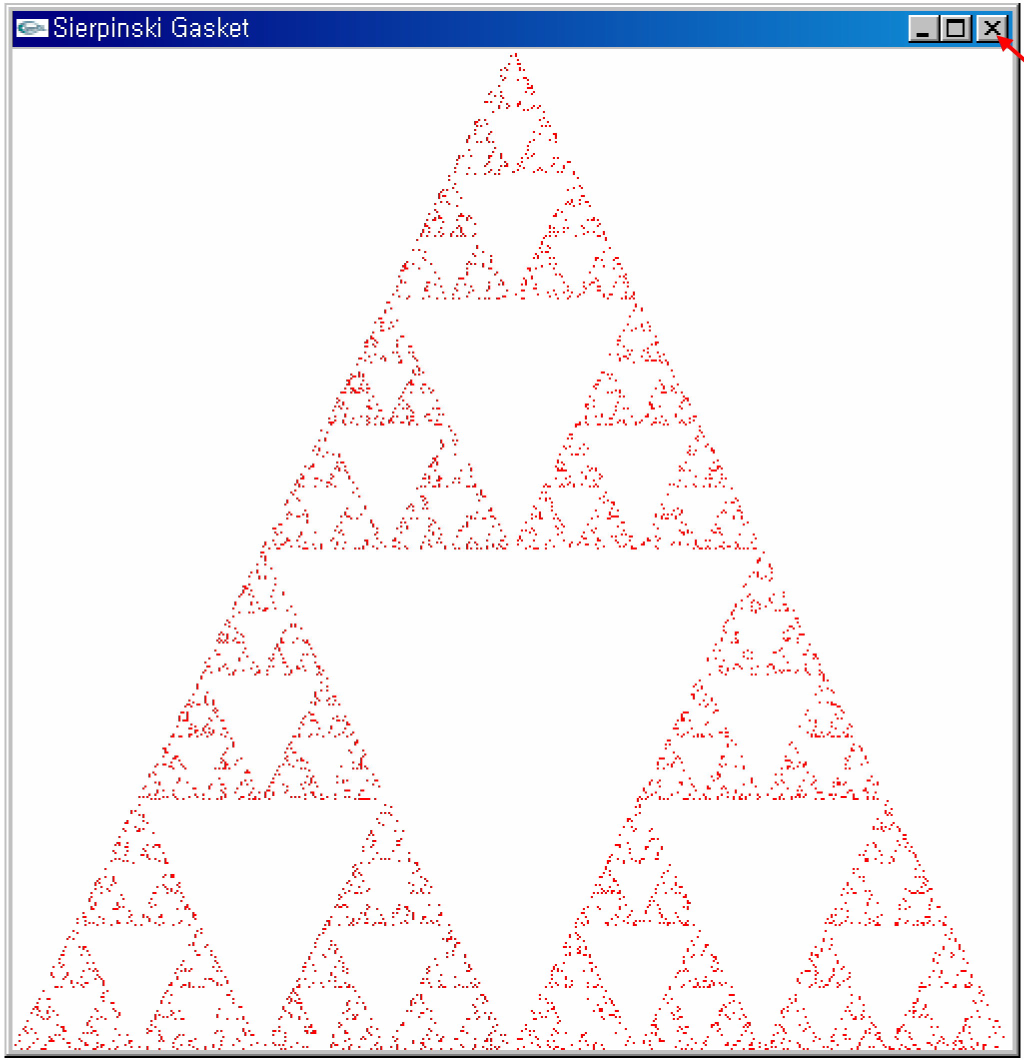
myInit function

```
void myInit(void) {
  /* attributes */
 glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
 glColor3f(1.0, 0.0, 0.0);            /* draw in red */

 /* set up viewing */
 /* 500 x 500 window with origin lower left */
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(0.0, 500.0, 0.0, 500.0);
 glMatrixMode(GL_MODELVIEW);
}
```

display function

```
void display( void ) {
   point2 vertices[3]={{0.0,0.0},{250.0,500.0},{500.0,0.0}}; /* A triangle */
   point2 p ={75.0,50.0};            /* An arbitrary initial point inside triangle */
   int j, k;

   glClear(GL_COLOR_BUFFER_BIT);  /*clear the window */
   for (k=0; k<5000; k++) {
      j=rand( ) % 3;     /* pick a vertex at random */
      /* Compute point halfway between selected vertex and old point */
      p[0] = (p[0] + vertices[j][0]) / 2.0;
      p[1] = (p[1] + vertices[j][1]) / 2.0;
      /* plot new point */
      glBegin(GL_POINTS);
         glVertex2fv(p);
      glEnd( );
   }
   glFlush( ); /* flush buffers */
}
```

실행 결과

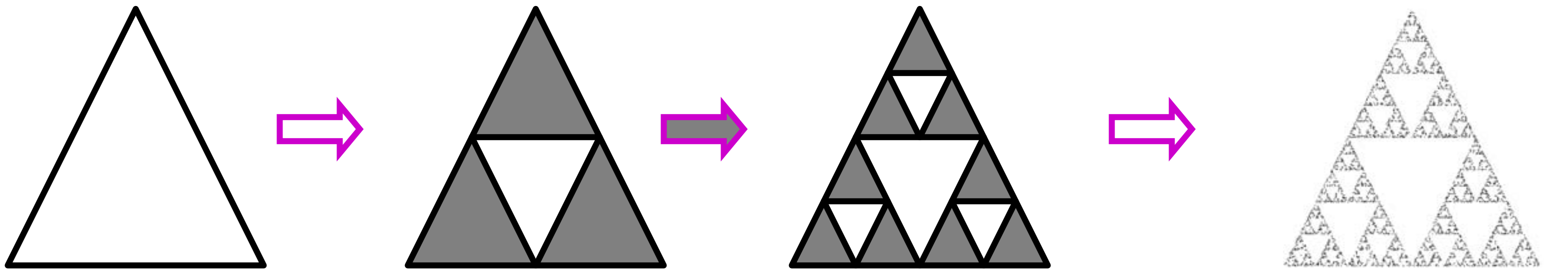


press to quit

2.8 Polygons and Recursion

Sierpinski Gasket, again

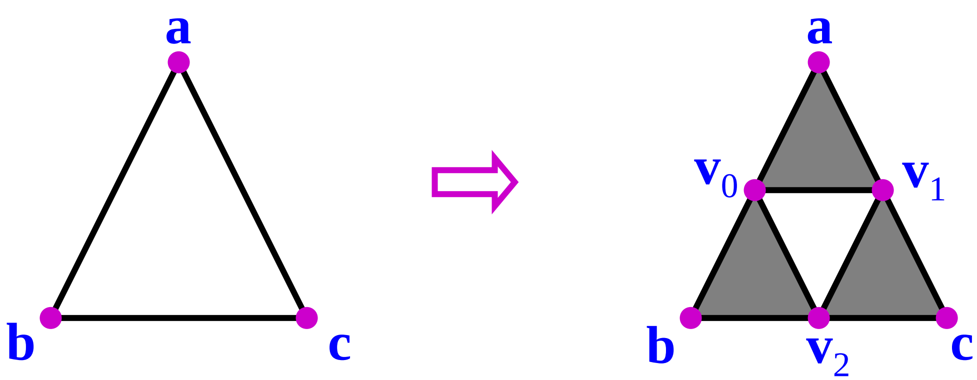
- Sierpinski gasket 의 특징
  – 각 삼각형의 가운데 ¼은 항상 비어있다



- another way of generating Sierpinski gasket ?
  – recursion

Recursion

- triangle의 subdivision



– 원래의 삼각형 : (**a**, **b**, **c**)

– midpoints 계산

$$\mathbf{v}_0 = \frac{\mathbf{a}+\mathbf{b}}{2}, \quad \mathbf{v}_1 = \frac{\mathbf{a}+\mathbf{c}}{2}, \quad \mathbf{v}_2 = \frac{\mathbf{b}+\mathbf{c}}{2}$$

– subdivided triangles : $(\mathbf{a}, \mathbf{v}_0, \mathbf{v}_1), (\mathbf{c}, \mathbf{v}_1, \mathbf{v}_2), (\mathbf{b}, \mathbf{v}_2, \mathbf{v}_0)$

OpenGL 구현 예제

```
#include <stdio.h>
#include <stdlib.h>

typedef float point2[2];                  /* 2D point */
point2 v[]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};     /* initial triangle */

void triangle( point2 a, point2 b, point2 c) {             /* display one triangle */
 glBegin(GL_TRIANGLES);
  glVertex2fv(a);
  glVertex2fv(b);
  glVertex2fv(c);
 glEnd();
}
```

OpenGL 구현 예제

```
void divide_triangle(point2 a, point2 b, point2 c, int m) {   /* triangle subdivision */
 point2 v0, v1, v2;
 int j;
 if (m > 0) {
  for(j=0; j<2; j++) v0[j]=(a[j] + b[j]) / 2;
  for(j=0; j<2; j++) v1[j]=(a[j] + c[j]) / 2;
  for(j=0; j<2; j++) v2[j]=(b[j] + c[j]) / 2;
  divide_triangle(a, v0, v1, m–1);
  divide_triangle(c, v1, v2, m–1);
  divide_triangle(b, v2, v0, m–1);
 } else
  triangle(a,b,c); /* draw triangle at end of recursion */
}

void display(void) {
 glClear(GL_COLOR_BUFFER_BIT);
  divide_triangle(v[0], v[1], v[2], n);
 glFlush();
}
```

$$v_x = \frac{a_x + b_x}{2}, v_y = \frac{a_y + b_y}{2}$$

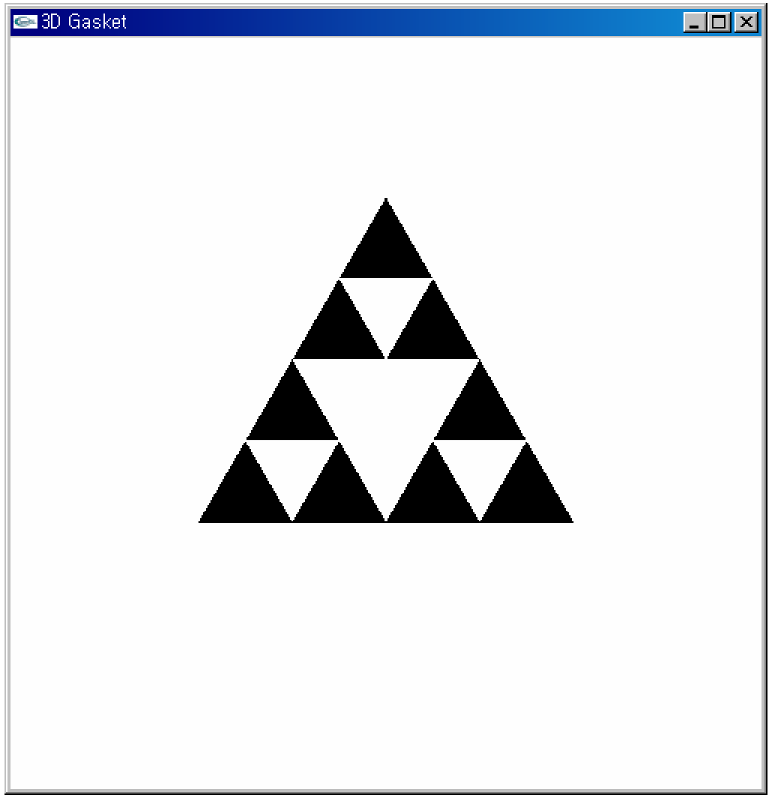
OpenGL 구현 예제

```
void myinit( ) {
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
  glMatrixMode(GL_MODELVIEW);
  glClearColor (1.0, 1.0, 1.0, 1.0);
  glColor3f(0.0,0.0,0.0);
}
```

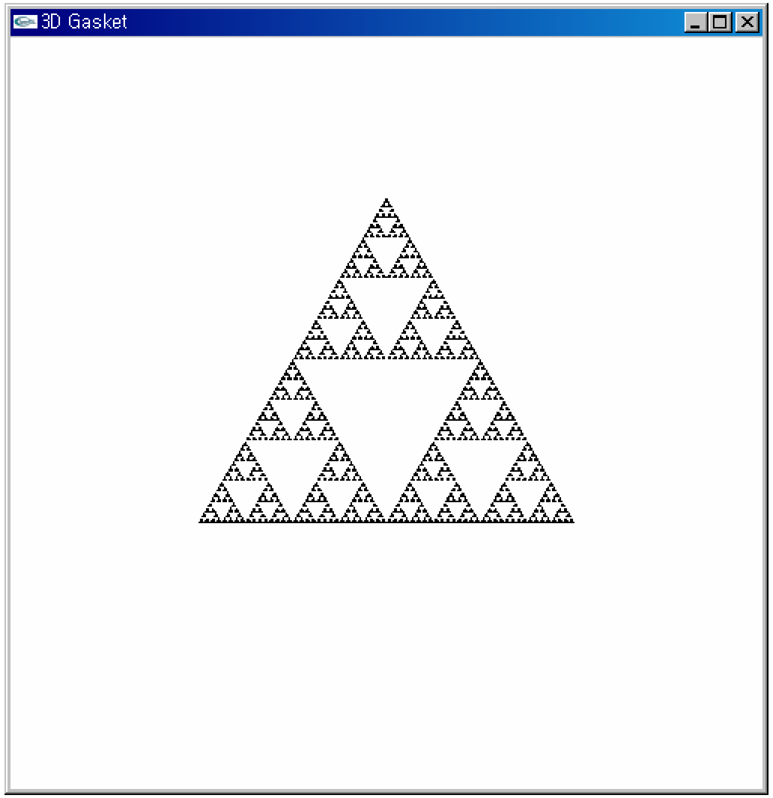
```
void main(int argc, char *argv[]) {
  if (argc != 2) {
    fprintf(stderr, "usage: %s number\n",
      argv[0]);    /* 종료 조건 필요 ! */
    exit(0);
  }
  n=atoi(argv[1]);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE |
      GLUT_RGB );
  glutInitWindowSize(500, 500);
  glutCreateWindow("3D Gasket");
  glutDisplayFunc(display);
  myinit();
  glutMainLoop();
}
```

실행 예제

- gasket2 2



- gasket2 6

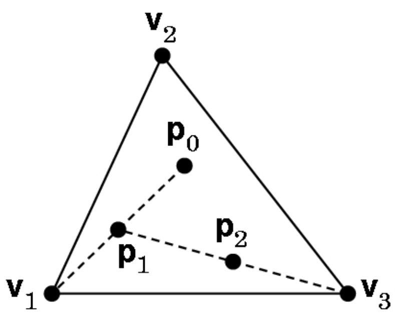


2.9 The Three-Dimensional Gasket

3D Sierpinski gasket

- 2D triangle → 3D tetrahedron 으로 확장

```
typedef struct { float x, y, z; } point;
point vertices[4] =
  {{0,0,0},{250,500,100},{500,250,250},{250,100,250}}; /* A tetrahedron */

point old_pt = {250,100,250};      /* start point */
point new_pt;                      /* new point */
```

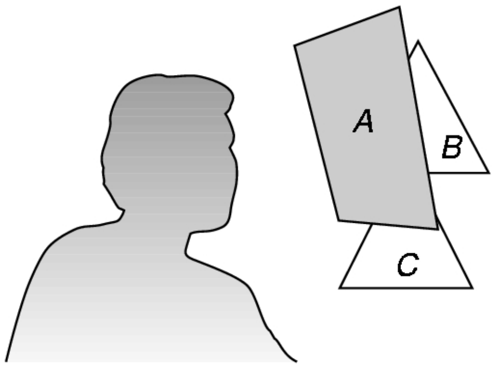


3D Sierpinski gasket

```
void display(void) {   /* computes and plots a single new point */
  int i;
  j = rand( ) % 4; /* pick a vertex at random */
  /* Compute point halfway between vertex and old point */
  new_pt.x = (old_pt.x + vertices[j].x) / 2;
  new_pt.y = (old_pt.y + vertices[j].y) / 2;
  new_pt.z = (old_pt.z + vertices[j].z) / 2;

  glBegin(GL_POINTS);           /* plot point */
    glColor3f(1.0-new_pt.z/250.,new_pt.z/250.,0.); /* 거리감을 위해서 색깔 구분*/
    glVertex3f(new_pt.x, new_pt.y,new_pt.z);
  glEnd();

  /* replace old point by new */
  old_pt.x=new_pt.x; old_pt.y=new_pt.y; old_pt.z=new_pt.z;
  glFlush();
}
```

Hidden Surface Removal

- 3D 확장 시의 근본적인 문제
  - 그리는 순서 대로 출력하면, 안 된다
  - camera를 기준으로 서로 가리는 관계를 반영해야



B, C는 A에 의해서 가려진다

- hidden surface removal algorithm
  - = visible surface algorithm
  - camera 에서 보이는 부분만 남기는 algorithm

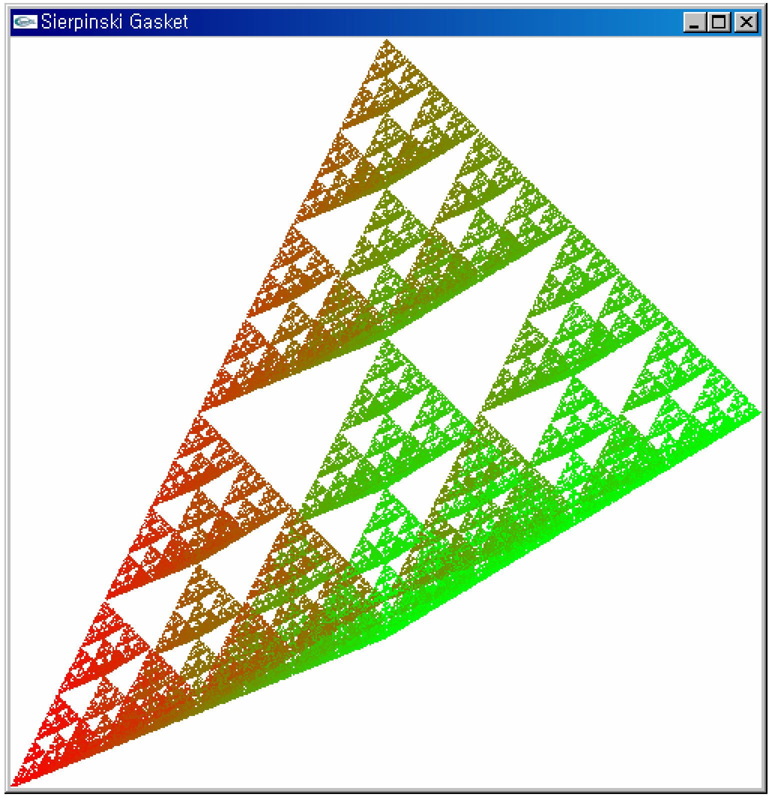
Z-buffer algorithm

- Z-buffer algorithm (= depth-buffer algorithm)
  - 가장 간단한 HSR algorithm
  - OpenGL에서는 기본적으로 제공

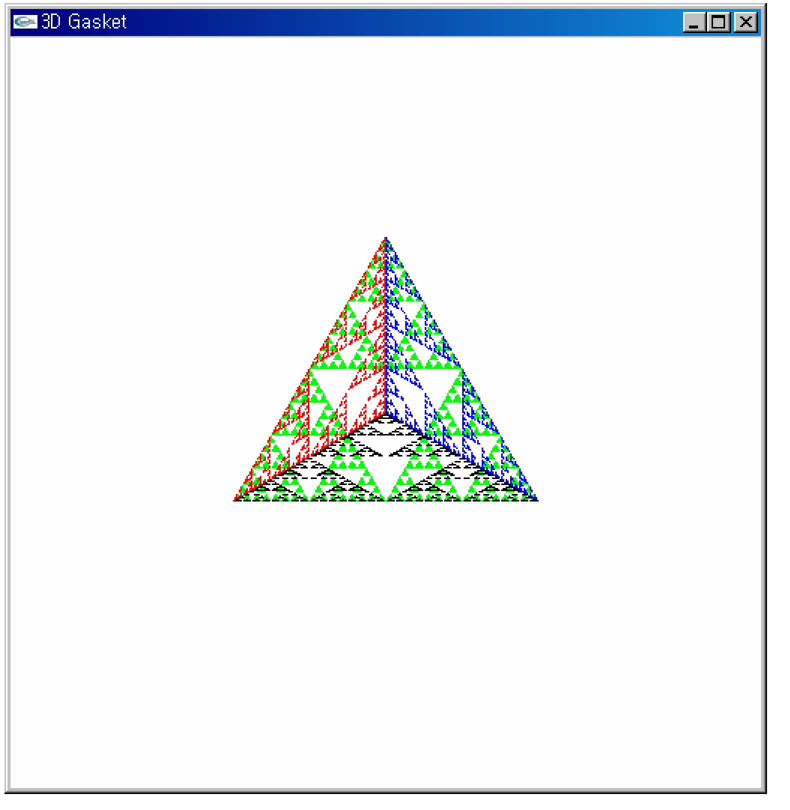
- void glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
  - Z-buffer 를 사용하기 위해 준비
- glEnable(GL_DEPTH_TEST);
  - Z-buffer를 on
- glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  - Z-buffer를 clear

실행 예제

- gasket3d.c : without Z-buffer
  - point 만 출력



- tetra.c : with Z-buffer
  - tetra 5



Suggested Readings

- OpenGL Architecture Review Board,
  OpenGL Programming Guide, 2^{nd} Ed.,
  Addison-Wesley, (1997).

- OpenGL Architecture Review Board,
  OpenGL Reference Manual, 2^{nd} Ed.,
  Addison-Wesley, (1997).

- Mark J. Kilgard,
  GLUT Specification, version 3,
  http://reality.sgi.com/opengl/spec3/spec3.html