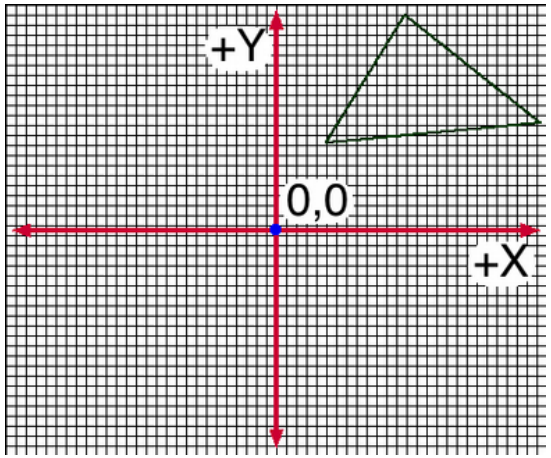
Graphics Programming

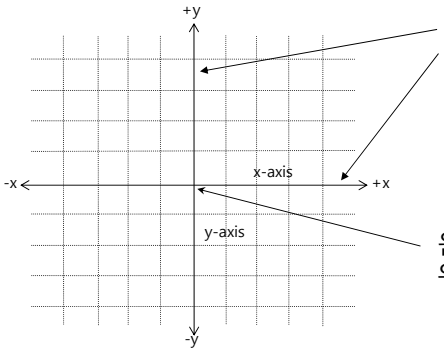
514780
2017년 가을학기
9/7/2017
단국대학교 박경신

Coordinate Systems



2D Cartesian Coordinate Systems

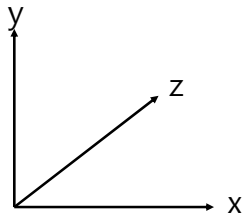
- Cartesian Coordination Systems



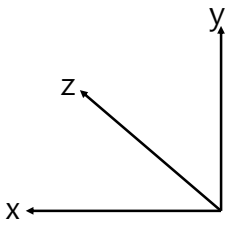
Two axes: **x-axis** and **y-axis**, two straight lines perpendicular to each other, both pass through origin and extends infinitely in two opposite directions

원점 (Origin)은 **좌표계**의 중심에 위치하고 있고 값은 (0, 0)이다.

3D Cartesian Coordinate Systems

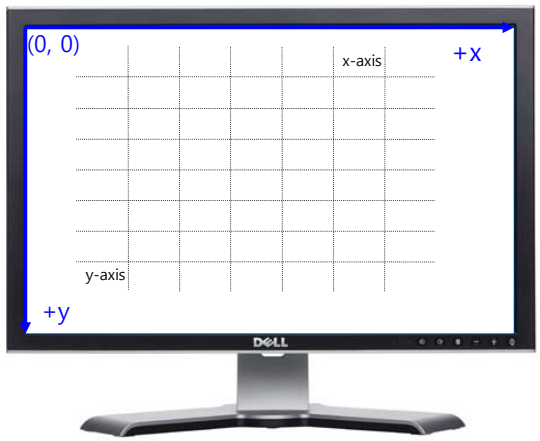


- 왼손 좌표계 (Left-handed coordinate system)는 x+는 오른쪽, y+는 위쪽, z+는 화면안쪽.



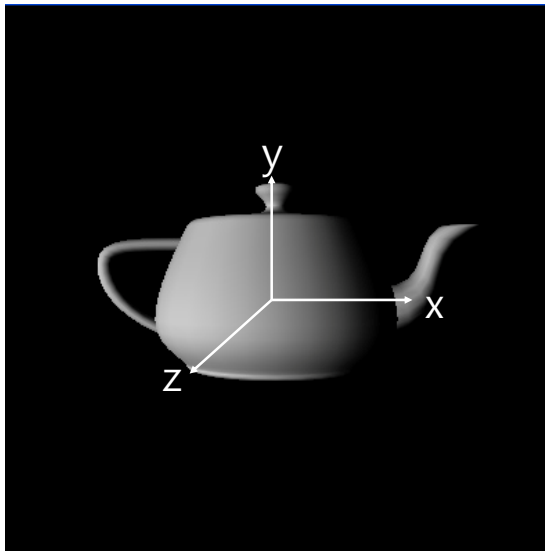
- 오른손 좌표계 (Right-handed coordinate system)는 x+는 왼쪽, y+는 위쪽, z+는 화면안쪽.

Screen Coordinate System



- Screen coordinate system은 원점(Origin)이 화면의 좌측상단에 위치하고 값은 (0, 0)이다. x+ 오른쪽. y+ 아래쪽.
- 1 unit = 1 pixel

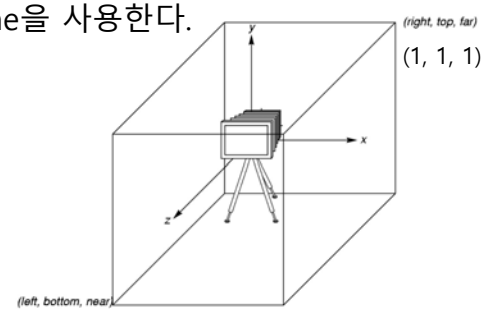
3D Coordinate Systems



- OpenGL은 오른손 좌표계 (Right-handed coordinate system)
- x+ 오른쪽. y+ 위쪽. z+ 화면 밖으로 나오는 방향.

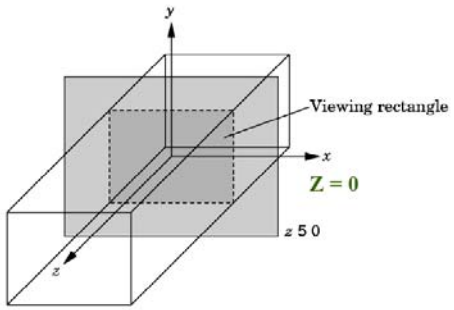
OpenGL Camera

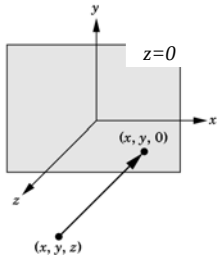
- OpenGL에서는 카메라가 물체의 공간(drawing coordinates)의 원점(origin)에 위치하며 z- 방향으로 향하고 있다.
- 관측공간을 지정하지 않는다면, 디폴트로 2x2x2 입방체의 viewing volume을 사용한다.



Orthographic Viewing

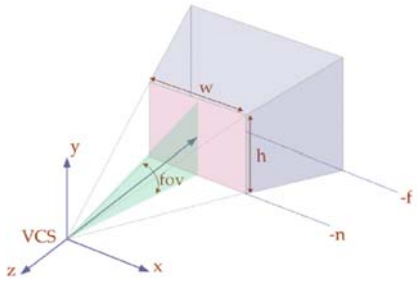
- 직교 투영 (Orthographic parallel projection)
  - Ortho(left, right, bottom, top, zNear, zFar);
  - 기본 직교 투영에서는 점들은 z-축을 향해 z=0 평면에 투영





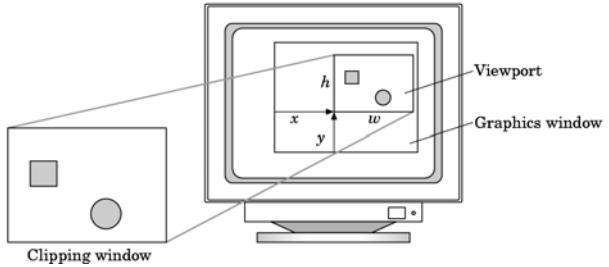
Perspective Viewing

- 원근 투영 (Perspective projection)
  - Frustum(left, right, bottom, top, zNear, zFar);
  - Perspective(fovy, aspect, zNear, zFar); - 상하좌우값을 설정하는 대신 y방향의 시선각도 (FOV)와 종횡비(가까운 쪽 클리핑 평면의 너비를 높이로 나눈 값)를 사용



Viewport Functions

- 뷰포트 (Viewport)
  - 윈도우 내부에 설정한 공간. 그리기가 뷰포트 내부로 제한됨.
- glViewport(x, y, width, height)
  - 윈도우를 처음 생성할 때 전체 윈도우에 해당하는 픽셀 영역을 뷰포트로 설정; 이보다 작은 영역을 뷰포트로 설정할 때는 glViewport() 사용. 일반적으로 윈도우 전체를 뷰포트로 사용.
  - GLUT Reshape function이 있을 경우, glViewport()가 반드시 포함되어야 함.

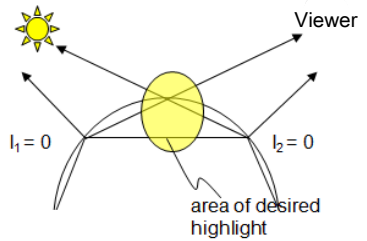


Transformations and Viewing

- OpenGL에서 projection matrix (transformation)를 사용하여 projection을 수행함
- Transformation 함수는 좌표계 변환을 위해 사용하였음
- 그러나 OpenGL 3.0 이전 transformation 함수들은 deprecated (더 이상 사용하지 않길 권고함)
- 3가지 선택
  - Application code
  - GLSL functions
  - GLM (OpenGL Mathematics) vector, matrix

Conventional OpenGL Rendering Pipeline

- OpenGL에서 지원하는 옵션과 상태 변수를 검사해서 적용여부를 판단하므로 저사양 HW에서는 비효율적
- Modified Phong Illumination Model만 지원하는 고정된 조명 계산
- Gouraud Shading만 지원하는 고정된 음영처리
  - 정점 색을 계산한 후 정점 색을 보간하여 픽셀 색을 결정
  - Mach Band가 나타나거나 픽셀 값이 잘못 계산될 수 있음

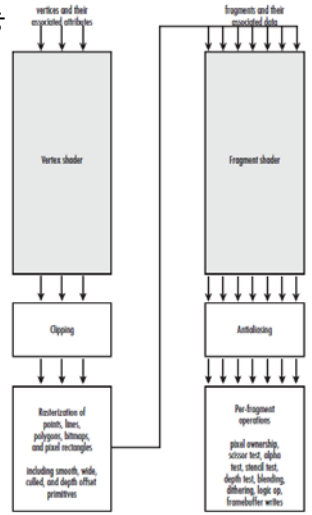


Extending OpenGL

- 그래픽 하드웨어의 발전에 따라 복잡한 그래픽 기법을 적용하기 위한 기능의 지원 필요
- OpenGL은 새로운 버전에 추가된 기능을 확장 기능으로 지원
  - 이전 버전의 API를 수정하지 않음으로써 이전 버전과의 호환성 유지
  - 함수나 매크로 상수 이름에 확장 기능을 식별할 수 있도록 접미어를 붙여 명명
    - _ARB, _EXT, _NV, _ATI 등등
- 프로그래머블 하드웨어를 지원하기 위한 API를 확장 기능으로 제공
  - 고정 파이프라인을 이용하는 대신 사용자가 작성한 코드대로 음영 처리를 할 수 있는 프로그래머블 파이프라인의 이용이 가능

Programmable Pipeline

- Vertex Shader, Fragment Shader를 작성하여 다양한 렌더링 기법을 적용 가능



OpenGL Shader

- 기본 Shaders
  - Vertex shader
  - Fragment shader

Vertex Shader Applications

- Moving vertices
  - Morphing
  - Wave motion
  - Fractals
- Lighting
  - More realistic models
  - Cartoon shaders

Fragment Shader Applications

- Per-fragment lighting calculations

per vertex lighting

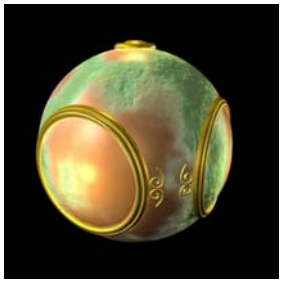
per fragment lighting

Fragment Shader Applications

- Texture mapping

Smooth shading

Environment mapping



Bump mapping

Simple Vertex Shader

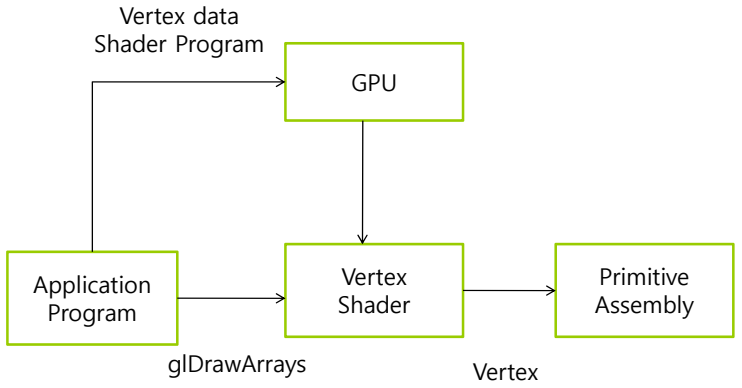
Input from application

```
in vec4 vPosition;
void main(void)
{
    gl_Position = vPosition;
}
```

Must link to variable in application

Built-in variable

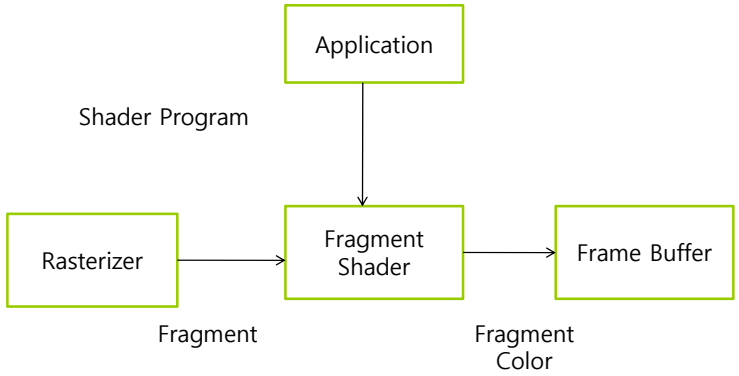
Execution Model



Simple Fragment Program

```
void main(void)
{
  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Execution Model



GLSL Data Types

- C types: int, float, bool
- Vectors: 벡터
  - float vec2, vec3, vec4
  - 또한 int (ivec)와 boolean (bvec)
- Matrices: mat2, mat3, mat4 행렬
  - 열 (columns) 우선으로 구성
  - 일반적인 참조방식은 m[row][column]
- Texture Sampler: 텍스쳐 접근이 가능한 샘플러 타입
  - sampler1D, sampler2D, sampler3D, samplerCube
  - sampler1DShadow, sampler2DShadow
- C++ style constructors
  - vec3 a = vec3(1.0, 2.0, 3.0)
  - vec2 b = vec2(a)

GLSL Pointers

- GLSL에는 pointer 개념이 없음
- C 언어의 구조체 (struct)을 이용해서 함수로 복사해서 사용가능
- Matrices나 Vectors 는 기본 형 (basic types)으로 GLSL 함수에 파라메터 입력이나 반환 형 출력으로 사용 가능

  mat3 func(mat3 a)

GLSL Qualifiers

- 변수 평가자 (Variable Qualifiers)
  - `const` – 상수
  - `attribute` – 전역 변수이며, **정점**마다 바뀔 수 있고, **OpenGL 프로그램**에서 Vertex Shader로 값을 변경함. 이 평가자는 Vertex Shader에서만 사용됨. 쉐이더에서는 읽기 전용.
    - Built-in vertex attribute: **gl_Position**
    - User-defined vertex attribute: **in vec3 velocity**
  - `uniform` – 전역 변수이며, **Primitive**마다 바뀔 수 있고, **OpenGL 프로그램**에서 쉐이더로 값을 변경함. 이 평가자는 Vertex Shader와 Fragment Shader모두에서 사용가능. 쉐이더에서 이 변수는 상수.
  - `varying` – Vertex Shader에서 Fragment Shader로 전달되는 변수. Vertex Shader에서는 쓰기가 허용되지만, Fragment Shader에서는 읽기 전용.
    - 최신버전에서는 Vertex Shader에서 out으로 쓰고, Fragment Shader에서 in으로 사용
    - User-defined varying variable: **out vec4 color;**

Example: Vertex Shader

```
const vec4 red = vec4(1.0, 0.0, 0.0, 1.0);
out vec3 color_out;
void main(void)
{
  gl_Position = vPosition;
  color_out = red;
}
```

Required Fragment Shader

```
in vec3 color_out;
void main(void)
{
  gl_FragColor = color_out;
}
// in latest version use form
// out vec4 fragcolor;
// fragcolor = color_out;
```

GLSL Operators and Functions

- 일반적인 C 함수
  - Trigonometric
  - Arithmetic
  - Normalize, reflect, length
- Overloading of vector and matrix types

```
mat4 a;
vec4 b, c, d;
c = b*a; // a column vector stored as a 1d array
d = a*b; // a row vector stored as a 1d array
```

GLSL Constructor

- 생성자 (Constructor)
  - 변수의 초기화는 C++ 생성자 방식을 이용
    - vec3 n = vec3(0.0, 1.0, 0.0);
  - 생성자는 초기화 외에 식에서도 사용가능
    - greenColor = myColor + vec3(0.0, 1.0, 0.0);
  - 벡터에 하나의 스칼라값을 지정하면 벡터의 모든 요소에 할당
    - ivec4 whiteColor = ivec4(255);
  - 스칼라와 벡터, 행렬을 생성자 내에서 혼합해 사용할 수 있고, 여분의 요소가 있는 경우 버려짐
    - vec4 v = vec4(x, vec2(y, z), w);
  - 행렬은 열 우선으로 구성되고, 단일 스칼라 값을 지정하는 경우 대각 행렬이 됨 (대각 이외의 요소는 0으로 채워짐)
    - mat2 m = mat2(1.0, 0.0, 0.0, 1.0);
    - mat2 m = mat2(1.0);
  - 형변환은 생성자를 통해서만 가능
    - float j = 4.7; int i = int(j);

GLSL Swizzling and Selection

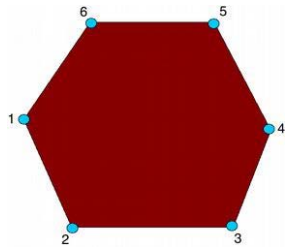
- [] 또는 (.) operator를 사용하여 벡터 및 행렬 요소에 접근
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q
  - vec3 a = vec3(0.0, 0.0, 1.0); **a[2], a.b, a.z, a.p**는 모두 다 같음
  - mat3 m = mat3(1.0); float element21 = m[2][1]; // 0.0
  - mat3 m = mat3(1.0); vec3 column1 = m[0]; // (1, 0, 0)
- 요소 선택자를 이용하여 재배치 및 복제 가능
  - vec3 myZYX = s.zyx;
- 요소 선택자를 사용하여 벡터 일부 요소만 수정 가능
  - vec4 a; a.yz = vec2(1.0, 2.0);

GLSL Passing Values

- 함수의 반환형으로 배열을 제외한 모든 타입이 사용 가능
- 함수의 인자로는 배열 및 구조체를 포함한 모든 타입이 사용 가능
- **Call by value**로만 호출되므로 다음 한정자를 사용하여 함수 내의 인자 값이 변경될 수 있는 여부를 지정할 수 있음
  - **in** (default)
  - **const in**
  - **out**
  - **inout** (deprecated)

OpenGL Geometry

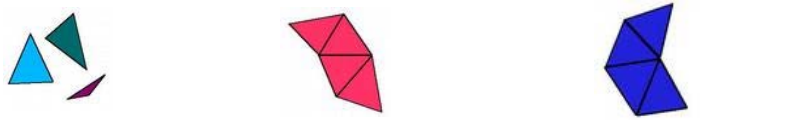
- 가상의 공간을 구성하는 각 물체를 표현하는데 있어 가장 기본이 되는 요소
- 실시간 그래픽스에서는 주로 가장 단순한 형태의 표현 방법인 linear primitives를 사용
  - Point, vertex
  - Line segments
  - Polygon
  - Polyhedron



OpenGL Geometry Primitives

GL_POINTS GL_LINES GL_LINE_STRIP GL_LINE_LOOP

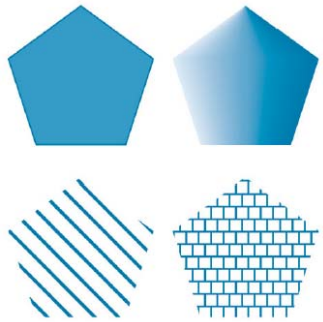
GL_TRIANGLES GL_TRIANGLE_STRIP GL_TRIANGLE_FAN



OpenGL Attributes

- 각 기하학적 기본요소 (geometry primitive)는 속성을 갖고 있다. 속성은 기본 요소가 화면상에 나타날 수 있는 방법을 제어한다.
  - Color
  - Line thickness
  - Line styles
  - Polygon patterns

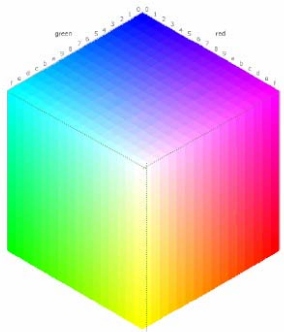
선의 두께나 스타일

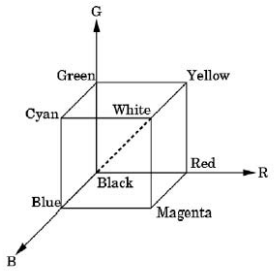


다각형 표시방법

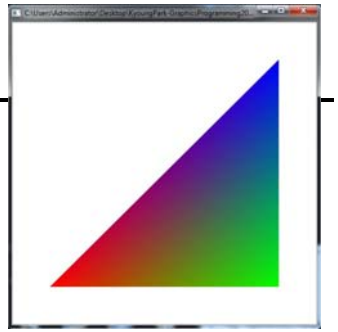
OpenGL Attributes

- OpenGL Color Model
  - RGB (Red Green Blue) or RGBA(Red Green Blue Alpha)
  - RGB 색이 따로 분리돼서 framebuffer에 저장되어 있음.





Color Triangle



```
const float vertexColor[] = { 1.0f, 0.0f, 0.0f, 1.0f, 0.0f,
1.0f, 0.0f, 1.0f, 0.0f,              0.0f, 1.0f, 1.0f };
const float vertexPositions[] = { -0.75f, -0.75f, 0.0f, 1.0f,
0.75f, -0.75f, 0.0f, 1.0f,           0.75f, 0.75f, 0.0f, 1.0f };
void SetData() {
    glGenVertexArrays(1, &vao);          // vao
    glBindVertexArray(vao);
    glGenBuffers(2, &vbo[0]);            // vbo
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);       // vertex position
    glBufferData(GL_ARRAY_BUFFER, 12*sizeof(GLfloat), vertexPositions, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);       // vertex color
    glBufferData(GL_ARRAY_BUFFER, 12*sizeof(GLfloat), vertexColor, GL_STATIC_DRAW);
    glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(1);
    glBindVertexArray(0);
}
```