OpenGL

2020 Computer Graphics

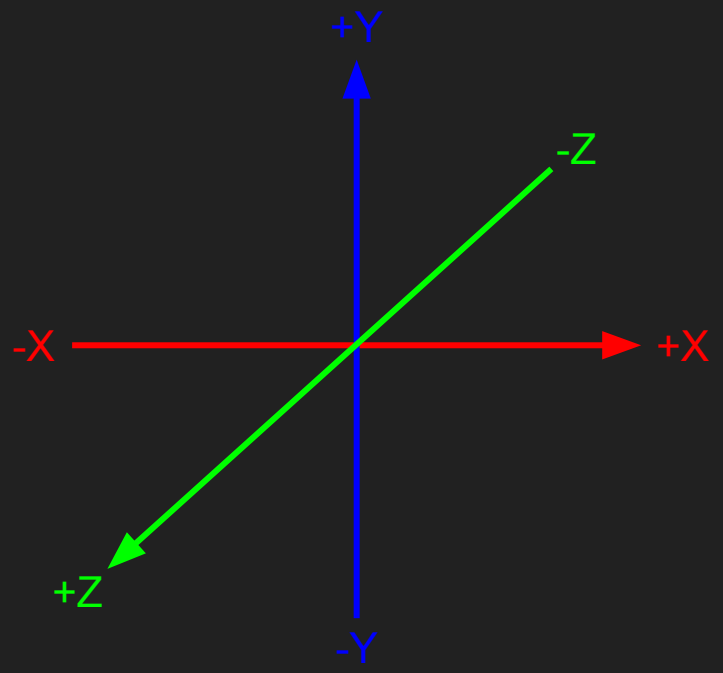
Document

https://www.khronos.org/registry/OpenGL-Refpages/gl4/

https://www.glfw.org/docs/latest/modules.html

OpenGL coordinate system

- Right-handed system



OpenGL datatype

suffix	data type	C/C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	Short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit float	Float	GLfloat, GLclampf
d	64-bit float	Double	GLdouble,GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte,GLboolean
us	16-bit unsigned number	unsigned short	GLushort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint,Glenum,GLbitfield

Initialization

- int glfwInit(void);
  - initialize the GLFW library
  - must be use before other glfw functions

- void glfwTerminate(void);
  - destroy all remaining windows and cursors, restore any modified gamma ramps and free any other allocated resources
  - if glfwInit() success, should be called before the application exits
  - if glfwInit() fails, there is no need to call this function, as it is called by glfwInit() before it returns failure

Initialization

- GLFWerrorfun glfwSetErrorCallback(GLFWerrorfun callback);
  - set the error callback
  - callback: new callback, or NULL to remove the currently set callback

Window

- GLFWwindow* glfwCreateWindow(int width, int height, const char * title, GLFWmonitor * monitor, GLFWwindow * share);
  - create a window, return NULL if an error occured
  - width, height: the desired width and height in screen coordinates, of the window.
  - title: window title
  - monitor: the monitor to use for full screen mode, or NULL for windowed mode.
  - share: the window whose context to share resources with, or NULL to not share resources

- void glfwDestroyWindow(GLFWwindow * window);
  - destroy the specified window and its context
  - window: the window to destroy

Window

- void glfwMakeContextCurrent(GLFWwindow * window);
  - make the OpenGL context of the specified window current on the calling thread
  - window: the window whose context to make current, or NULL to detach the context

- int glfwWindowShouldClose(GLFWwindow * window);
  - returns the value of the close flag of the specified window
  - window: the window to query

Window

- GLFWframebuffersizefun glfwSetFramebufferSizeCallback (GLFWwindow * window, GLFWframebuffersizefun callback);
  - set the framebuffer resize callback of the specified window
  - is called when the framebuffer of the specified window is resized
  - window: the window whose callback to set.
  - callback: the new callback, or NULL to remove the currently set callback
- void glfwGetFramebufferSize(GLFWwindow * window, int * width, int * height);
  - retrieves the size, in pixels, of the framebuffer of the specified window
  - window: the window whose framebuffer to query
  - width, height: store the width and height in pixels, of the framebuffer, or NULL

Input

- GLFWkeyfun glfwSetKeyCallback(GLFWwindow * window, GLFWkeyfun callback);
  - set the key callback of the specified window
  - is called when a key is pressed, repeated or released

- GLFWmousebuttonfun glfwSetMouseButtonCallback(GLFWwindow * window, GLFWmousebuttonfun callback);
  - set the mouse button callback of the specified window
  - is called when a mouse button is pressed or released
  - window: the window whose callback to set
  - callback: the new key callback, or NULL to remove the currently set callback

Input

- int glfwGetKey(GLFWwindow * window, int key);
  - returns the last state reported for the specified key to the specified window
  - returned state is GLFW_PRESS or GLFW_RELEASE
  - window: the desired window
  - key: the desired keyboard key

- void glfwGetCursorPos(GLFWwindow * window, double * xpos, double * ypos);
  - return the position of the cursor, in screen coordinates, relative to the upper-left corner of the content area of the specified window
  - window: the desired window
  - xpos: store the cursor x-coordinate, or NULL
  - ypos: store the cursor y-coordinate, or NULL

Input

- double glfwGetTime(void);
  - return the current GLFW time, in seconds
  - unless the time has been set using glfwSetTime it measures time elapsed since GLFW was initialized

- void glfwPollEvents(void);
  - process only those events that are already in the event queue and then return immediately
  - processing events will cause the window and input callbacks associated with those events to be called

Color representation

- RGBA: red, green, blue, alpha
  - each channel has intensity from 0.0~1.0
  - values outside this interval will be clamp to 0.0 or 1.0
  - alpha is used in blending and transparency

- void glColor{34}{sifd}[v](...);
  - specify a color
    - glColor3f(1.0f, 0.0f, 0.0f);
    - glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
    - glColor3fv(float*);

Color buffer

- void glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);
  - red, green, blue, alpha: Specify the red, green, blue, and alpha values used when the color buffers are cleared

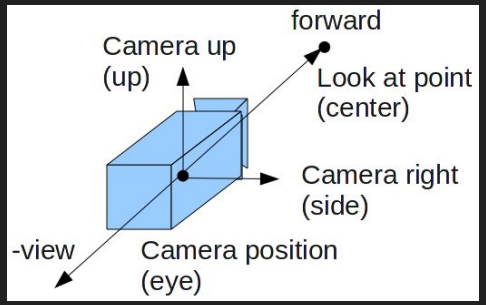
- void glClear(GLbitfield mask);
  - clear the specified buffers to their current clearing values
  - mask: GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT, GL_ACCUM_BUFFER_BIT, GL_STENCIL_BUFFER_BIT
    - glClear(GL_COLOR_BUFFER_BIT);

Depth buffer & Depth test

- void glEnable(GLenum cap);
  - enable or disable GL capabilities
  - cap: GL_DEPTH_TEST, GL_STENCIL_TEST, …
- void glClearDepth(GLdouble depth);
  - depth: specifies the depth value used when the depth buffer is cleared
- void glDepthFunc(GLenum func);
  - specify the depth comparison function
  - func: GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS (default: GL_LESS)
- glClear(GL_DEPTH_BUFFER_BIT);
  - glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

ModelView matrix

- void glMatrixMode(GLenum mode);
  - switch between three modes: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE
  - each matrix mode has its own matrix stack
- void glLoadIdentity(void);
  - replace the current matrix with the identity matrix
- void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdoubpe upz);
  - eyex, eyey, eyez: where the camera is positioned
  - centerx, centery, centerz: where the camera looks at
  - upx, upy, upz: the up-vector of the camera

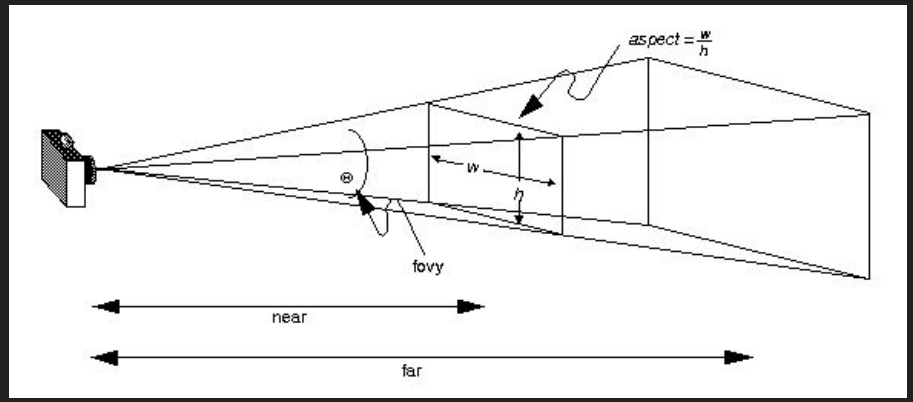


Projection matrix

- glMatrixMode(GL_PROJECTION);
- glLoadIdentity();
- We are really concerned with only two types of projection transformations:
  - Orthographic projection: Our viewing volume is rectangular and all objects appear the same size no matter the distance.
  - Perspective projection: Uses perspective to give a sense of depth. Our viewing volume is conical or pyramidal in shape.

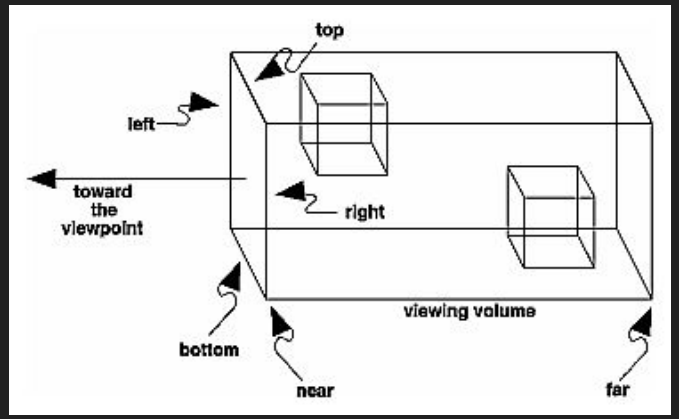
Projection matrix

- void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
  - fovy: the field of view angle, in degrees, in the y direction.
  - aspect: the aspect ratio that determines the field of view in the x direction
  - zNear: the distance from the viewer to the near clipping plane (positive)
  - zFar: the distance from the viewer to the far clipping plane (positive)



Projection matrix

- void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
  - left, right: the coordinates for the left and right vertical clipping planes
  - bottom, top: the bottom and top horizontal clipping planes
  - near, far: the distances to the nearer and farther depth clipping planes (can be negative)
  - void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
    - equal to calling glOrtho with near = 1 and far = 1



Viewport transformation

- void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
  - transform the final image into some region of the window
  - x, y: the lower-left corner of the viewport rectangle, in pixels
    - default : (0, 0)
  - width, height: the width and height of the viewport
    - default is set to the dimensions of that window

Basic transformation

- void glPushMatrix(void);
  - push current matrix into matrix stack

- void glPopMatrix(void);
  - Pop matrix from matrix stack

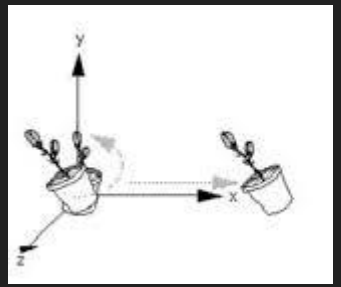
- These stack operations of matrix is very useful for constructing a hierarchical structure.

Basic transformation

- void glTranslate{fd}(TYPE x, TYPE y, TYPE z);
  - multiplies current matrix by a matrix that moves an object by (x, y, z)
  - TYPE: GLfloat or GLdouble
  - x, y, z: specify the x, y, and z coordinates of a translation vector

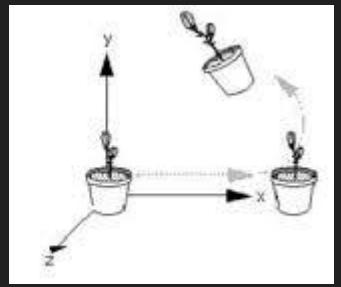
- void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);
  - multiplies current matrix by a matrix that rotates an object in a counterclockwise direction about the ray from origin to (x, y, z) with angle as the degrees
  - TYPE: GLfloat or GLdouble
  - rotation follows the right-hand rule

Basic transformation



gltranslatef(1, 0, 0);
glrotatef(45.0, 0, 0, -1);
drawObject();

$$[T][R]\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



glrotatef(45.0, 0, 0, -1);
gltranslatef(1, 0, 0);
drawObject();

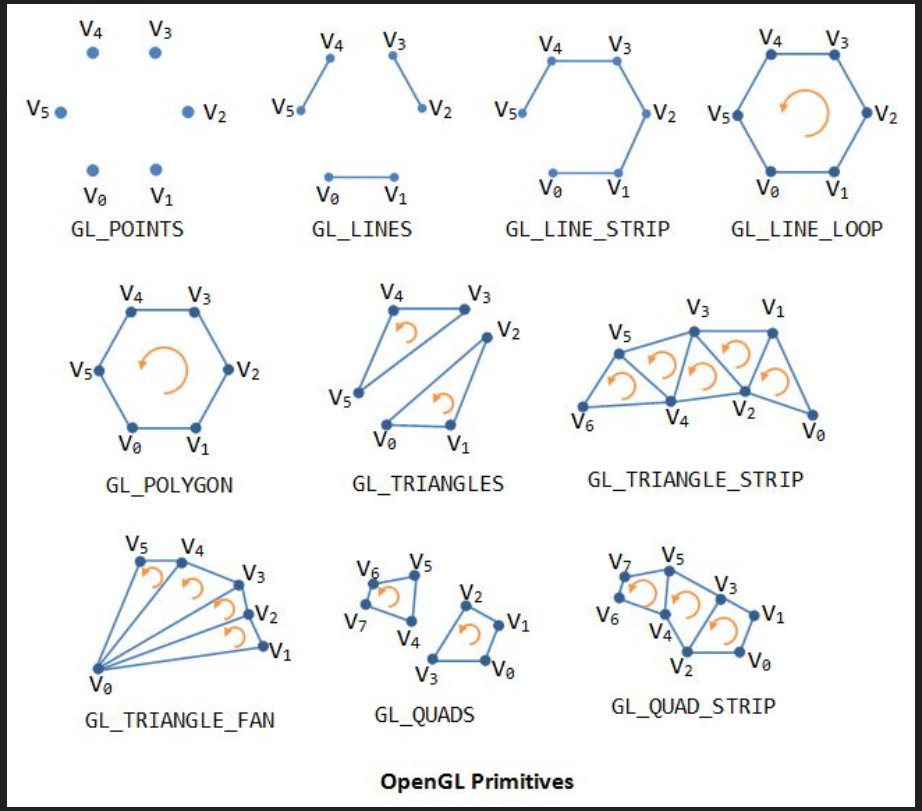
$$[R][T]\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Point, line and polygon

- void glBegin(GLenum mode);
  - marks the beginning of a vertex-data list
  - vertex-data include vertex's color, normal, position, texcoord, etc.
  - mode: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_POLYGON, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUAD, GL_QUAD_STRIP

- void glEnd(void);
  - marks the end of a vertex-data list

Point, line and polygon



Point, line and polygon

- void glColor{34}{sifd}[v](...);

- void glNormal3{bsifd}[v](...);
  - set the current normal vector
  - normal vectors must be normalize: glEnable(GL_NORMALIZE);

- void glVertex{234}{sifd}[v](...);
  - specify a vertex for use in describing a geometric object
  - only effective between a glBegin() and glEnd() pair
  - set vertex's attributes before glVertex
    - use glColor and glNormal before glVertex to set the vertex's color

Face culling

- glEnable(GL_CULL_FACE);

- void glCullFace(GLenum mode);
  - specify whether front- or back-facing facets can be culled
  - mode: GL_FRONT, GL_BACK, and GL_FRONT_AND_BACK (default: GL_BACK)

- void glFrontFace(GLenum mode);
  - define front- and back-facing polygons
  - mode: GL_CW, GL_CCW
  - default: GL_CCW

Completion of drawing

- void glfwSwapBuffers(GLFWwindow * window);
  - swap the front and back buffers of the specified window when rendering with OpenGL
  - window: the window whose buffers to swap

Lighting

- glEnable(GL_LIGHTING);
- glEnable(GL_LIGHTi);
  - i = 0 ~ GL_MAX_LIGHT-1
  - at least eight lights are supported in OpenGL (GL_MAX_LIGHT ≥ 8)
- void glLight[fi][v](GLenum light, GLenum pname, const GLfloat* param)
  - light: specify a light.
  - pname: specify a light source parameter for light
    - GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION, GL_SPOT_DIRECTION
  - param: specify the value that parameter pname of light source light will be set to
  - ex. glLightfv(GL_LIGHT0, GL_POSITION, x, y, z, w);
    - if w = 0, directional light

Texture

- glEnable(GL_TEXTURE_2D);
- void glGenTextures(GLsizei n, GLuint* textures);
  - generate texture name(id)
  - n: the number of texture names to be generated
  - textures: an array in which the generated texture names are stored
- void glBindTexture(GL_TEXTURE_2D, GLuint texture);
  - bind a named texture to a texturing target before using or setting it
  - texture: the name(id) of a texture
- glBindTexture(GL_TEXTURE_2D, 0);
  - unbind texture objects if you don’t want to use them on next objects

Texture

- void glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const void * data);
  - specify a two-dimensional texture image
  - target: GL_TEXTURE_2D, GL_TEXTURE_1D_ARRAY, ...
  - level: specify the level-of-detail number (level 0 is the base image level)
  - internalformat: specify the number of color components in the texture
    - GL_RGB, GL_RGBA, GL_DEPTH_COMPONENT…
  - width, height: specify the width and height of the texture image
  - border: must be 0
  - format: specify the format of the pixel data ex. GL_RGB, GL_RGBA, …
  - type: specify the data type of the pixel data ex. GL_UNSIGNED_BYTE
  - data: specify a pointer to the image data in memory

Texture

- void glTexParameter{fi}[v](GL_TEXTURE_2D, GLenum pname, TYPE param);
  - set texture parameters
  - pname: specify the texture parameter
  - param: specify the value of pname
  - https://learnopengl.com/Getting-started/Textures

pname	param
GL_TEXTURE_WRAP_S GL_TEXTURE_WRAP_T GL_TEXTURE_WRAP_R	GL_REPEAT, GL_MIRRORED_REPEAT, GL_CLAMP_TO_EDGE, GL_MIRROR_CLAMP_TO_EDGE, GL_CLAMP_TO_BORDER.
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR GL_NEAREST_MIPMAP_NEAREST GL_LINEAR_MIPMAP_NEAREST GL_NEAREST_MIPMAP_LINEAR GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_MAG_FILTER	GL_NEAREST, GL_LINEAR

Texture

- void glTexCoord{1234}{sifd}[v](TYPE coordinate);
  - assign texture coordinate for each vertex
  - coordinate value: 0~1
    - glTexCoord2f(u, v);
    - glVertex3f(x, y ,z);