CMT107 Visual Computing

II.1 Introduction to OpenGL

Xianfang Sun

School of Computer Science & Informatics
Cardiff University

Overview

- Introduction to OpenGL
  - What is OpenGL
  - OpenGL History
  - OpenGL Pipeline
  - OpenGL Components
  - Java OpenGL (Jogl)
    - Installation of Jogl on Eclipse
- OpenGL Programming
  - Basic OpenGL Coding Framework
  - OpenGL Geometric Primitives
  - A Simple OpenGL Program
    - In C
    - In Java

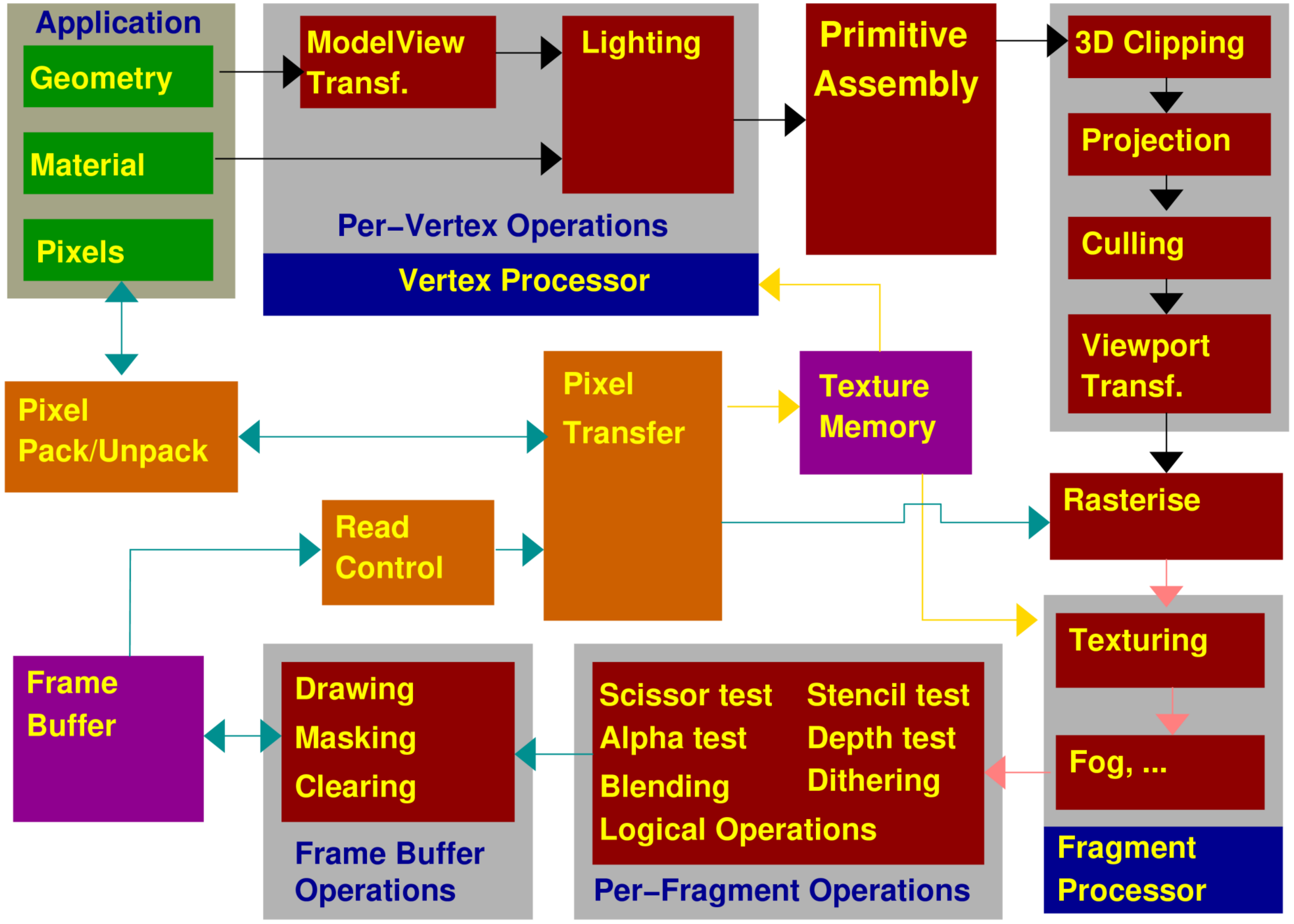
What is OpenGL?

- OpenGL: Open Graphics Library
  - Originally IRIS GL (Integrated Raster Imaging System Graphics Library) from Silicon Graphics
- OpenGL is NOT a language, it is
  - a software interface to graphics hardware
  - a graphics programming library
  - a standard for 3D graphics
- At the lowest possible level it still allows device independence
  - OpenGL is partly implemented in software and partly in hardware depending on the device
  - No high-level modelling operations, etc.

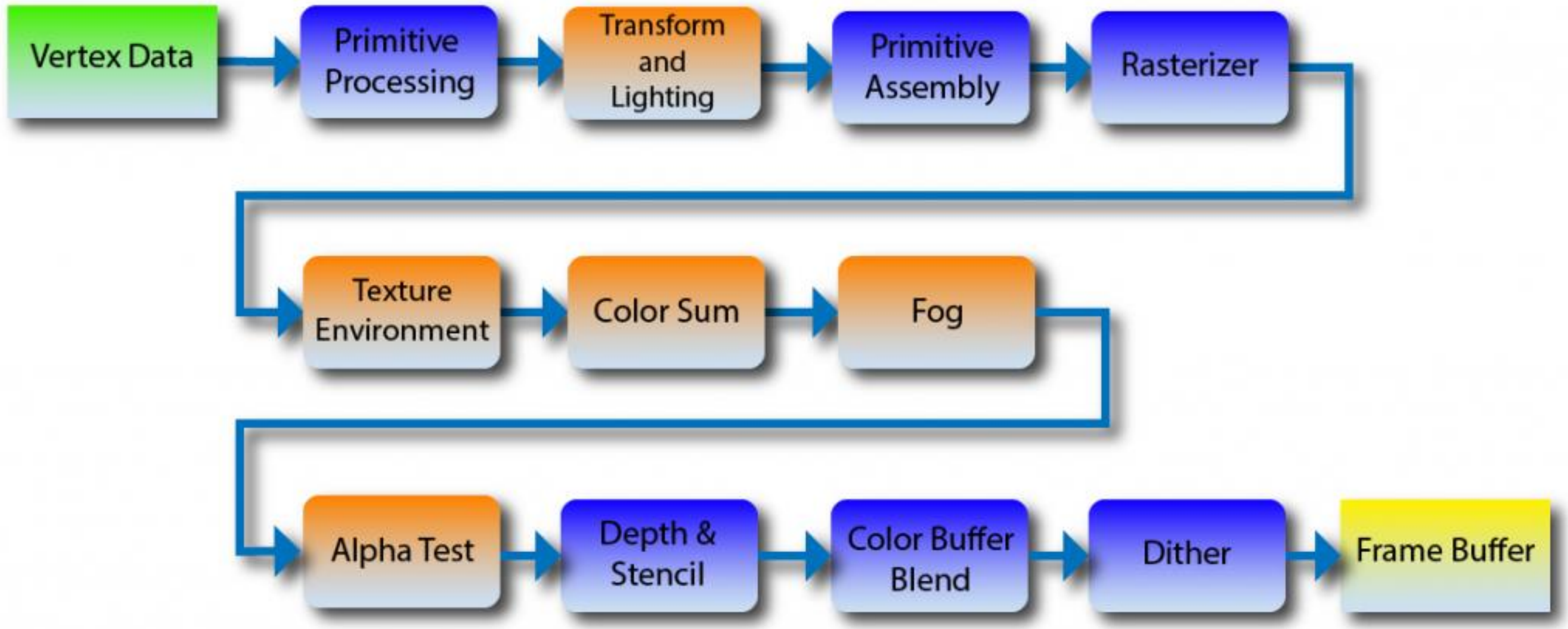
OpenGL History

OpenGL Release	GLSL Release	Year	Features
1.0	---	1992	Fixed-function Pipeline
1.1 ~ 1.5	---	1997 ~ 2003	
2.0	1.10	2004	vertex shaders and fragment shaders
2.1	1.20	2006	
3.0 ~ 3.2	1.30 ~ 1.50	2008 ~ 2009	Deprecated features; Geometry shaders from 3.2.
3.3	3.30	2010	
4.0, 4.1	4.00, 4.10	2010	Tessellation shaders
4.2	4.20	2011	
4.3	4.30	2012	Compute shaders
4.4	4.40	2013	
4.5	4.50	2014	
4.6	4.60	2017	

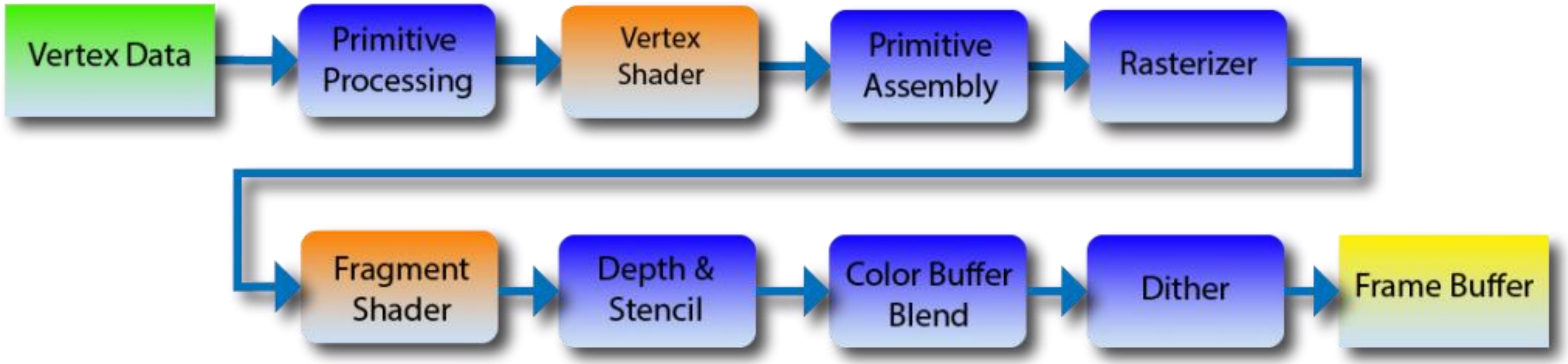
The OpenGL Pipeline



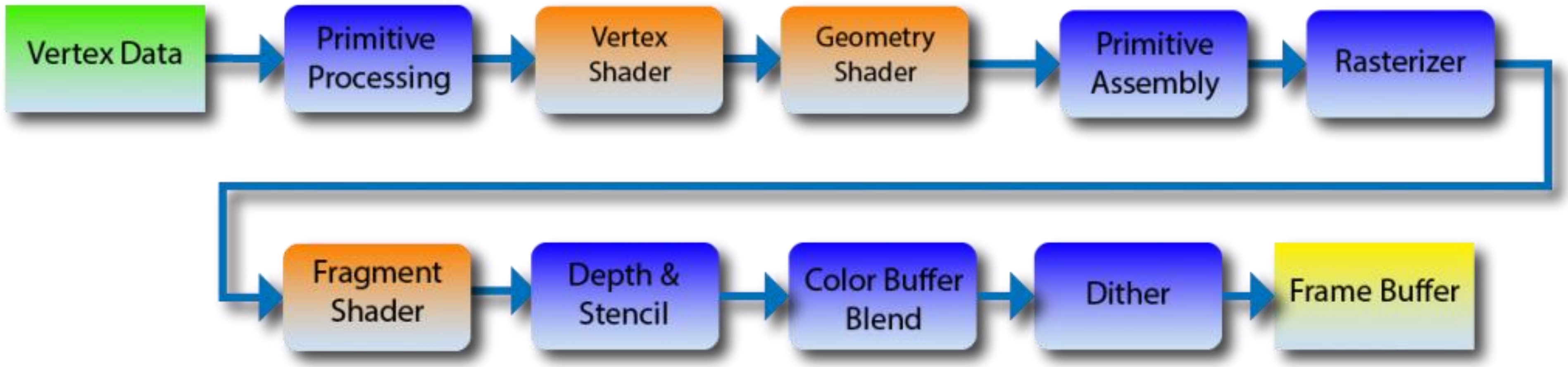
The OpenGL Pipeline (Ver < 2.0)



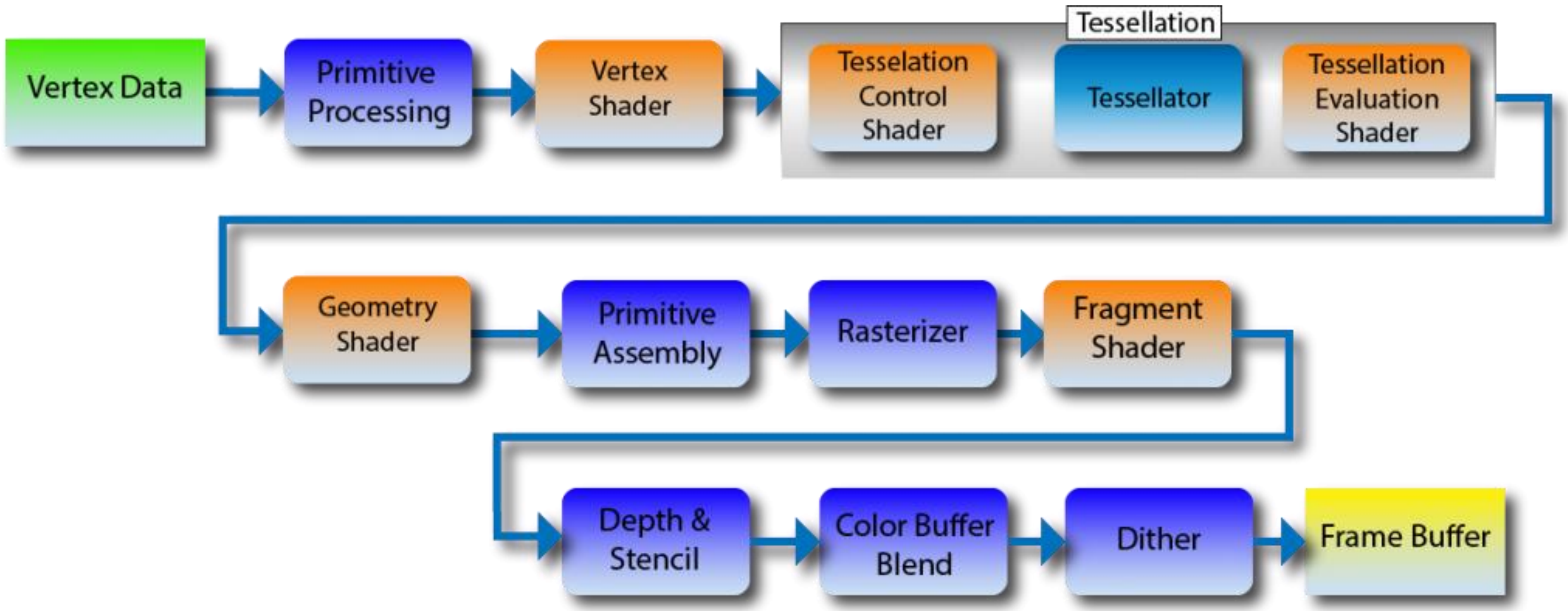
The OpenGL Pipeline (Ver = 2.0)



The OpenGL Pipeline (Ver = 3.2)

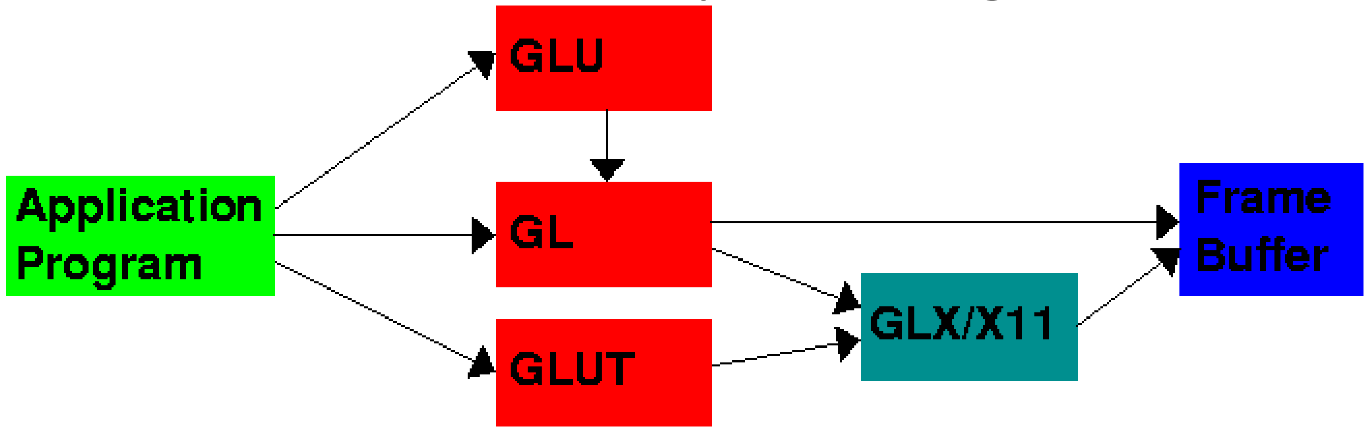


The OpenGL Pipeline (Ver = 4.0)



OpenGL Components

- Components of the OpenGL interface:
  - GL: core OpenGL functions
  - GLU: graphics utility library
    (a variety of graphics accessory functions, e.g. gluLookAt)
  - GLUT: OpenGL Utility Toolkit
    (interface to windowing system via xlib; alternatives: glib+GTK, QT; helpers for creating common objects, e.g. spheres, the teapot)
  - GLX: low-level interface to X11
    (different interfaces for other platforms: glw for windows)



Java OpenGL (JOGL)

- Java OpenGL (JOGL) is a wrapper library that allows OpenGL to be used in the Java programming.
- JOGL 1.1.1 gives full access to the APIs in the OpenGL 2.0 specification and limited access to GLU NURBS, providing rendering of curved lines and surfaces via the traditional GLU APIs.
- JOGL 2.0 provides full access to the APIs in the OpenGL 1.3 - 3.0, 3.1 - 3.3, ≥ 4.0, ES 1.x and ES 2.x specification as well as nearly all vendor extensions.
- Newest version (2.3.2) of JOGL can be downloaded from http://jogamp.org/deployment/jogamp-current/archive/

Installation of Jogl on Eclipse

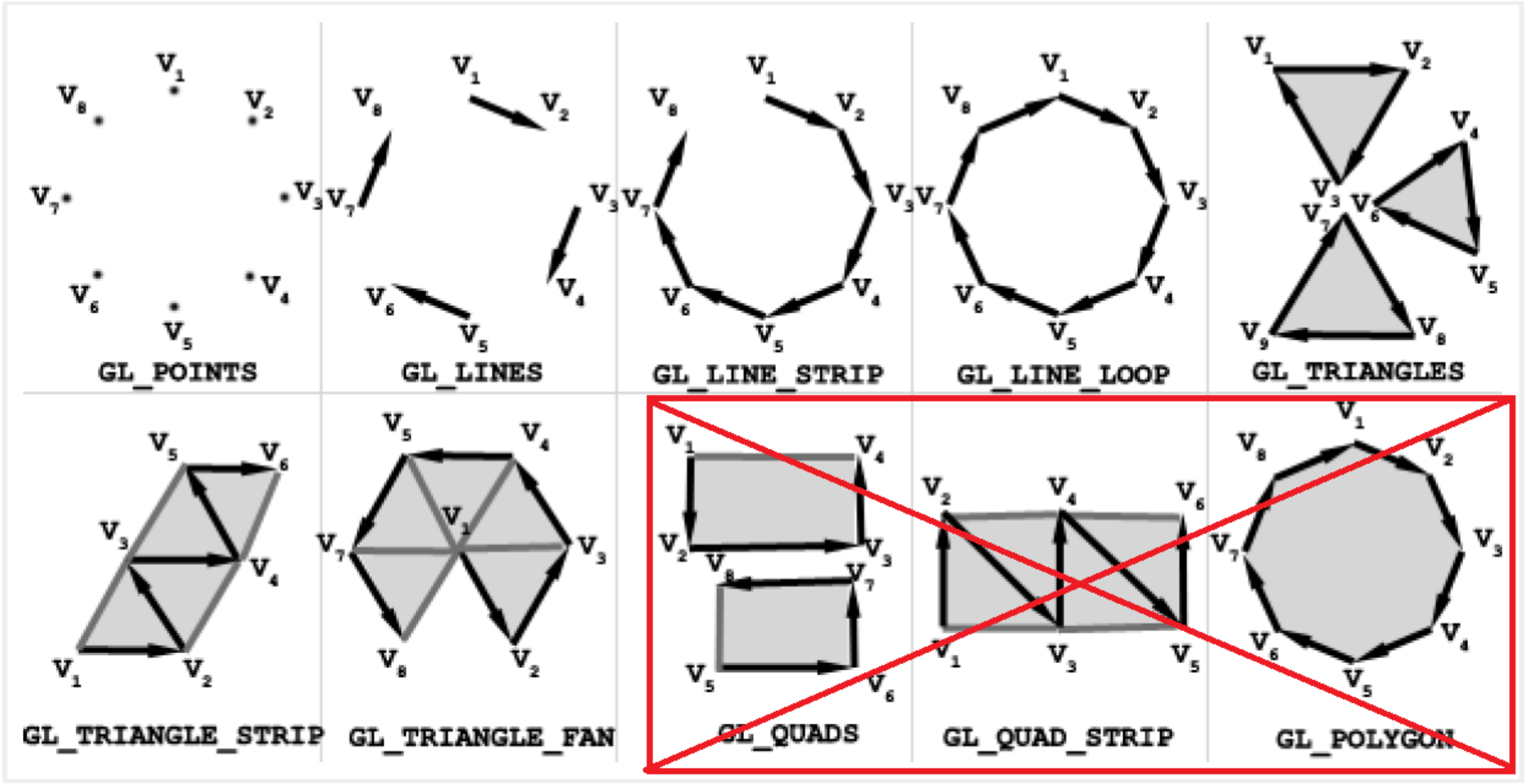
- Download and install Eclipse.
- Download and install the latest Jogl api.
- Set up Jogl as a user library.
- Configure Jogl library in each OpenGL (Jogl) project.

- All downloads and install instruction are free available from related official sites.
- More detail about installation can be found in the file available from learningcentral.

Basic OpenGL Coding Framework

- Configure OpenGL
  - Create window, Display mode
- OpenGL state initialisation
  - Set background colour, View positions, ……
  - Compile and link shader programs
- Set up Display Function
  - Render the scene
- Set up Reshape Function
  - resize the view window and recompute projection matrices
- Process Event loop

OpenGL Geometric Primitives



A Simple OpenGL C Program (1)

```c
#include <GL/glew.h>
#include <GL/freeglut.h>
// Define: number of Vertex Array Objects,
// number of Vertex Buffer Objects,
// number of Vertices

const GLuint numVAOs = 1, numVBOs = 1;
const GLuint numVertices = 1;

// Specify the ids of points, buffers,
// and the vertex attribute position
// in the vertex shader program.
GLuint idPoint = 0, idBuffer = 0;
GLuint vPosition = 0;

// Declare VAOs and VBOs
GLuint VAOs[numVAOs];
GLuint VBOs[numVBOs];
```

A Simple OpenGL C Program (2)

```
// Define: Vertex shader program, and Fragment shader program
const GLchar* srcVShader =
        "#version 330 core\n"
        "layout(location = 0) in vec4 vPosition;"
        "void main()"
        "{"
        "        gl_Position = vPosition;"
        "};";

const GLchar* srcFShader =
        "#version 330 core\n"
        "out vec4 fColor;"
        "void main()"
        "{"
        "        fColor = vec4(1.0, 0.0, 0.0, 1.0);"
        "}";
```

A Simple OpenGL C Program (3)

```
void init(void) // initialisation
{
   //Define vertices coordinates
  GLfloat vertices[numVertices][2] = {
          {0.0f, 0.0f}
   };

   //Generate vertex array objects (VAOs), and
   //Bind a VAO, i.e., initialise this VAO.
   // A second binding is needed later to use it
   glGenVertexArrays(numVAOs, VAOs);
   glBindVertexArray(VAOs[idPoint]);

   //Generate vertex buffer objects (VBOs), and
   //Bind a VBO, i.e., initialise this VBO.
   glGenBuffers(numVBOs, VBOs);
   glBindBuffer(GL_ARRAY_BUFFER, VBOs[idBuffer]);
   //The Data is then pooled into the buffer
   glBufferData(GL_ARRAY_BUFFER, sizeof(vertices),
                  vertices, GL_STATIC_DRAW);
  //Specify the location and data format of the
  //array of vertex attributes for rendering
   glVertexAttribPointer(vPosition, 2, GL_FLOAT,
                  GL_FALSE, 0, (void*)(0));
   glEnableVertexAttribArray(vPosition);
```

A Simple OpenGL C Program (4)

```
    //Create a shader program
    GLuint program = glCreateProgram();

    //Compile and attach vertex shader
    //into the program
    GLuint shader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(shader, 1, &srcVShader, NULL);
    glCompileShader(shader);
    glAttachShader(program, shader);
    glDeleteShader(shader);

    //Compile and attach fragment shader
    //into the program
    shader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shader, 1, &srcFShader, NULL);
    glCompileShader(shader);
    glAttachShader(program, shader);
    glDeleteShader(shader);

    //Link and use the shader program
    glLinkProgram(program);
    glUseProgram(program);
}
```

A Simple OpenGL C Program (5)

```c
// display the scene
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glPointSize(5);

    //Bind VAO again to use it
    glBindVertexArray(VAOs[idPoint]);
    glDrawArrays(GL_POINTS, 0, numVertices);

    glutSwapBuffers();
}

// resize the view window,
// and recompute projection matrices
void reshape(int width, int height){};
```

A Simple OpenGL C Program (6)

```c
int main(int argc, char** argv) {
    // Initialise GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE );
    glutInitWindowSize(512, 512);
    // Create display window
    glutCreateWindow(argv[0]);
    // OpenGL Version and profile
    glutInitContextVersion(3, 3);
    glutInitContextProfile(GLUT_CORE_PROFILE);

    // Deal with OpenGL extensions issues
    glewExperimental = GL_TRUE;
    if( GLEW_OK != glewInit() )
        exit(EXIT_FAILURE);

    init();

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop(); // Start GLUT event loop
    return 0;
}
```

A Simple OpenGL Java Program (1)

```
// Import some packages
import java.nio.FloatBuffer;

import javax.swing.JFrame;

import com.jogamp.opengl.GL3;
import com.jogamp.opengl.GLAutoDrawable;
import com.jogamp.opengl.GLEventListener;
import com.jogamp.opengl.awt.GLCanvas;

// Import GL constant
import static com.jogamp.opengl.GL3.*;
```

A Simple OpenGL Java Program (2)

```
public class Simple extends JFrame implements GLEventListener
{
        private final GLCanvas canvas; // Declare a canvas
        // Define: number of Vertex Array Objects,
        // number of Vertex Buffer Objects,
        // number of Vertices
        // Specify the ids of points, buffers,
        // and the vertex attribute position
        // in the vertex shader program
        private int idPoint = 0, numVAOs = 1;
        private int idBuffer = 0, numVBOs = 1;
        private int vPosition = 0;
        private final int numVertices = 1;

       // Declare VAOs and VBOs
        private int[] VAOs = new int[numVAOs];
        private int[] VBOs = new int[numVBOs];
```

A Simple OpenGL Java Program (3)

```
private String[] srcVShader =
          { "#version 330 core \n"
          + "layout(location = 0) in vec4 vPosition;"
          + "void main()"
          + "{"
          + "      gl_Position = vPosition;"
          + "}" };

private String[] srcFShader =
          { "#version 330 core\n"
          + "out vec4 fColor;"
          + "void main()"
          + "{"
          + "      fColor = vec4(1.0, 0.0, 0.0, 1.0);"
          + "}" };
```

A Simple OpenGL Java Program (4)

```
public Simple() {
        canvas = new GLCanvas(); // Define the canvas
        // Listen for openGL events
        canvas.addGLEventListener(this);

        // Add the canvas into the windows frame
        add(canvas, java.awt.BorderLayout.CENTER);
        // Exit when click close
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(512, 512); // set the window size
        setTitle("Simple Graphics"); // window title
        setVisible(true); // Display the frame
}
```

A Simple OpenGL Java Program (5)

```
public void init(GLAutoDrawable drawable) {
        // Get the GL pipeline object
        GL3 gl = drawable.getGL().getGL3();

        //Define the vertex coordinates
        float[] vertexArray = { 0.0f, 0.0f };
        //wrap the vertex array into a FloatBuffer.
        FloatBuffer vertices = FloatBuffer.wrap(vertexArray);

        // Generate vertex array objects (VAOs), and
        // Bind a VAO, i.e., initialise this VAO.
        // A second binding is needed later to use it
        gl.glGenVertexArrays(numVAOs, VAOs, 0);
        gl.glBindVertexArray(VAOs[idPoint]);

        // Generate vertex buffer objects (VBOs), and
        // Bind a VBO, i.e., initialise this VBO.
        // The Data is then pooled into the buffer
        gl.glGenBuffers(numVBOs, VBOs, 0);
        gl.glBindBuffer(GL_ARRAY_BUFFER, VBOs[idBuffer]);
        gl.glBufferData(GL_ARRAY_BUFFER, vertexArray.length *
                (Float.SIZE / 8),vertices, GL_STATIC_DRAW);

        gl.glVertexAttribPointer(vPosition,2,GL_FLOAT,false,0,0L);
        gl.glEnableVertexAttribArray(vPosition);
```

A Simple OpenGL Java Program (6)

```
        // Create a shader program
        int program = gl.glCreateProgram();

        // Compile and attach vertex shader into the program
        int shader = gl.glCreateShader(GL_VERTEX_SHADER);
        gl.glShaderSource(shader, 1, srcVShader, null);
        gl.glCompileShader(shader);
        gl.glAttachShader(program, shader);
        gl.glDeleteShader(shader);

        // Compile and attach fragment shader into the program
        shader = gl.glCreateShader(GL_FRAGMENT_SHADER);
        gl.glShaderSource(shader, 1, srcFShader, null);
        gl.glCompileShader(shader);
        gl.glAttachShader(program, shader);
        gl.glDeleteShader(shader);

        // Link and use the shader program
        gl.glLinkProgram(program);
        gl.glUseProgram(program);
    }
```

A Simple OpenGL Java Program (7)

```
public void display(GLAutoDrawable drawable) {
        GL3 gl = drawable.getGL().getGL3();

        gl.glClear(GL_COLOR_BUFFER_BIT);
        gl.glPointSize(5);

        gl.glDrawArrays(GL_POINTS, 0, numVertices);

        }

public void reshape(GLAutoDrawable drawable, int x, int y,
                    int width, int height) {
        }

public void dispose(GLAutoDrawable drawable) {
        }

public static void main(String[] args) {
                JFrame f = new Simple();
        }
```

Summary

- ➢ What is the underlying model for the OpenGL library?
  - What are the components of OpenGL?
- ➢ Basic OpenGL programming with C++ or Java.
  - Describe the OpenGL coding framework.

CMT107 Visual Computing

II.2 Vectors and Matrices

Xianfang Sun

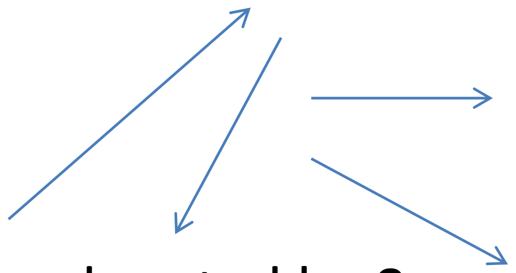
School of Computer Science & Informatics
Cardiff University

Overview

- Vectors
  - Vector Operations
  - Vector Geometry
  - Vector Projection
- 3D Vectors
  - Cross Product
  - 3D Vector Geometry
- Matrices
  - Special Matrices
  - Matrix Operations
  - Determinant

Vectors

- A vector is a directed line segment, characterised by:
  - Length
  - Direction
  - But NOT Position



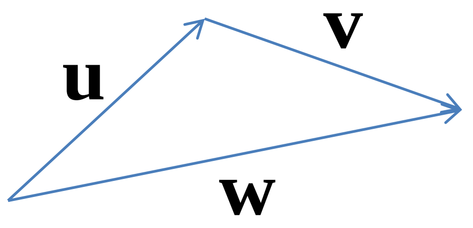
- A vector with length 0 is a zero vector, denoted by **0**
  - Zero vectors doesn't have direction.
- A vector with length 1 is a unit vector.
- A vector **u** with the same length but opposite direction of vector **v** is the negative vector of **v**, denoted by **u** = -**v.**
- Two vectors are equal iff they have the same length and the same direction.
  - Two zero vectors are always equal, though their directions are undefined.

Vector Operations

- A vector **u** multiplied by a scalar α denoted by α**u** has the same direction of **u** if $\alpha > 0$ and the opposite direction if $\alpha < 0$. The length of α**u** is $|\alpha|$ times of the length of **u**.
- The sum **w** of two vectors **u** and **v**:

  w = u + v

  follows the head-to-tail rule. That is,



  if the head of **u** is connected to the tail of **v**, then **w** is the directed line segment from the tail of **u** to the head of **v**.
- The subtraction of vector **v** from vector **w** is the addition of vector **w** and vector -**v**

  u = w – v = w + (-v)

Vector Operations

- A *n*D vector is represented by:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad \text{or} \quad \mathbf{v} = [v_1\, v_2 \cdots v_n]^T$$

- The sum and subtraction of two vectors are:

$$\mathbf{u} + \mathbf{v} = [u_1 + v_1\, u_2 + v_2 \cdots u_n + v_n]^T$$

$$\mathbf{u} - \mathbf{v} = [u_1 - v_1\, u_2 - v_2 \cdots u_n - v_n]^T$$

- The multiplication of a vector **v** by a scalar λ is defined by

$$\lambda\mathbf{v} = [\lambda v_1\, \lambda v_2 \cdots \lambda v_n]^T$$

- The inner product (dot product, scalar product) of two vectors is:

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

Vector Geometry

- A vector has direction and length.
  - The length is defined by

$$|\mathbf{v}| = \sqrt{\mathbf{v}\cdot\mathbf{v}} = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

  - The direction is parallel to the direction from the origin to the point $(v_1, v_2, \cdots, v_n)$ in *n*D Euclidean space.
  - The angle θ between two vectors **u** and **v** is calculated by

$$\cos\theta = \frac{\mathbf{u}\cdot\mathbf{v}}{|\mathbf{u}||\mathbf{v}|}$$

- Normalisation of a vector **v** gives a unit vector **v'**, which has length 1:

$$\mathbf{v}' = \mathbf{v}/|\mathbf{v}|$$

- Vectors **u** and **v** are orthogonal if $\mathbf{u}\cdot\mathbf{v} = 0$, which means that they are perpendicular to each other, and the angle between these two vectors are 90°.

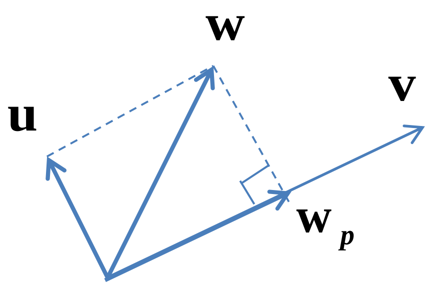
Vector Projection

- The vector projection $\mathbf{w}_p$ of a vector **w** on a nonzero vector **v** is a vector parallel to **v**, defined by

$$\mathbf{w}_p = \alpha\mathbf{v}$$

where α is a scalar calculated by

$$\alpha = \frac{\mathbf{w}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}}$$



- The vector **w** then can be represented by the sum of $\mathbf{w}_p$ and vector **u**, which is perpendicular to **v** (and $\mathbf{w}_p$).

$$\mathbf{w} = \mathbf{w}_p + \mathbf{u}$$

$$\mathbf{u}\cdot\mathbf{v} = 0, \quad \mathbf{u}\cdot\mathbf{w}_p = 0$$

Cross Product

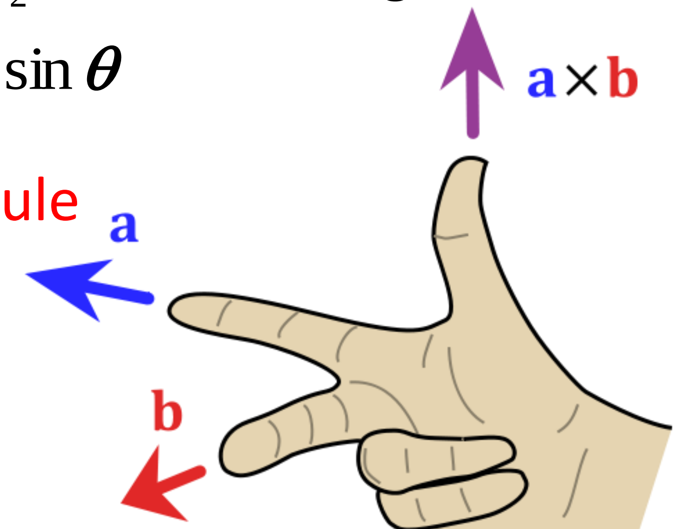
- Denote two 3D vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ by $\mathbf{v}_i = [x_i, y_i, z_i]^T$, the vector product (also called cross product, outer product) of $\mathbf{v}_1$ and $\mathbf{v}_2$ is defined by

$$\mathbf{v}_1 \times \mathbf{v}_2 = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}$$

- If θ is the angle between $\mathbf{v}_1$ and $\mathbf{v}_2$, then the length is:

$$|\mathbf{v}_1 \times \mathbf{v}_2| = |\mathbf{v}_1||\mathbf{v}_2| \sin \theta$$

- The direction satisfies right-hand rule
  - $\mathbf{v}_1 \times \mathbf{v}_2$ is perpendicular to both $\mathbf{v}_1$ and $\mathbf{v}_2$.



http://en.wikipedia.org/wiki/File:Right_hand_rule_cross_product.svg

3D Vector Geometry

- A point in 3D space can be represented by a 3D vector:

$$\boldsymbol{p} = [\boldsymbol{x}\ \boldsymbol{y}\ \boldsymbol{z}]^T$$

- A directed line segment from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$ can be represented by vector:

$$\boldsymbol{v} = \boldsymbol{p}_2 - \boldsymbol{p}_1$$

- Let $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are two directed line segments on a plane, then the normal direction is determined by the cross product of $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$:

$$\boldsymbol{n} = \boldsymbol{v}_1 \times \boldsymbol{v}_2$$

Matrices

- A matrix is a rectangular array of scalars, arranged in rows and columns. The individual items in a matrix are called its elements or entries. The number of rows and columns are referred to as the row and column dimensions.
- The following matrix **A** has row dimension *m* and column dimension *n*, or simply, *m* x *n* dimension. a_{ij} is an element of the matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

- The transpose of an *m* x *n* matrix **A**, denoted by $\mathbf{A}^T$, is the *n* x *m* matrix obtained by interchanging the rows and columns of A.
- To save the space, the matrix is often written as A = $[a_{ij}]_{mxn}$, or simply, A = $[a_{ij}]$, if the dimension of the matrix is implicitly known.

Special Matrices

- A square matrix is a matrix which has the same row and column dimension.
- A symmetric matrix is a square matrix that is equal to its transpose. Let **A**=[a_{ij}] be a symmetric matrix, then **A** = **A**^{T}. Its elements satisfy

$$\boldsymbol{a}_{ij} = \boldsymbol{a}_{ji}$$

- A diagonal matrix is a matrix (usually square matrix) in which the elements outside the main diagonal are all zero, *i.e.*, **A**=[a_{ij}],

$$\boldsymbol{a}_{ij} = 0, \quad \text{if } \boldsymbol{i} \neq \boldsymbol{j}$$

- A identity matrix, denoted by **I**, is a square diagonal matrix with 1's on the diagonal and 0's elsewhere

$$\boldsymbol{I} = [\boldsymbol{a}_{ij}] \;, \quad \boldsymbol{a}_{ij} = \begin{cases} 1 & \text{if } \boldsymbol{i} = \boldsymbol{j}; \\ 0 & \text{otherwise.} \end{cases}$$

Special Matrices

- A row matrix is a matrix of dimension 1 x n. It is also called a row vector.

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

- A column matrix is a matrix of dimension m x 1, also called a column vector.

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \qquad \mathbf{a} = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}^T$$

Matrix Operations

- Scalar-Matrix multiplication is defined by multiplying each element by the scalar
  - $\alpha\mathbf{A} = [\alpha a_{ij}]$
- Matrix-Matrix Addition of two matrices of the same dimension is defined by adding corresponding elements of the two matrices
  - $\mathbf{C} = \mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}]$
- Matrix-Matrix Multiplication of an m x l dimensional matrix **A** and an l x n dimensional matrix **B** is defined by
  - $\mathbf{C} = \mathbf{AB} = [c_{ij}]$
  - Where $c_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}$
- Inverse of a Square Matrix **A** is a square matrix **B**, such that

$$\mathbf{AB} = \mathbf{I}$$

  - Denote by $\mathbf{B} = \mathbf{A}^{-1}$

Orthogonal Matrix

- An orthogonal matrix is a square matrix with real entries whose columns and rows are orthogonal unit vectors (i.e., orthonormal vectors).
- Equivalently, a matrix **Q** is orthogonal if its transpose is equal to its inverse:

$$\mathbf{Q}^{-1} = \mathbf{Q}^{T}$$

which entails

$$\mathbf{Q}\mathbf{Q}^{T} = \mathbf{Q}^{T}\mathbf{Q} = \mathbf{I}$$

Determinant

- The determinant is a value associated with a square matrix, denoted by det(**A**), det **A**, or |**A**|. It is defined as

$$|\mathbf{A}| = \sum_{j=1}^{n} (-1)^{i+j} \boldsymbol{a}_{ij} |\mathbf{A}_{ij}|$$

  - where **A**$_{ij}$ is the (i, j) minor matrix of **A**, which is obtained by deleting the *i*th row and the *j*th column of **A**.

- The determinant of a 2 x 2 matrix is calculated by

$$|\mathbf{A}| = \begin{vmatrix} \boldsymbol{a}_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{21} & \boldsymbol{a}_{22} \end{vmatrix} = \boldsymbol{a}_{11}\boldsymbol{a}_{22} - \boldsymbol{a}_{12}\boldsymbol{a}_{21}$$

- The determinant of a 3 x 3 matrix is calculated by

$$|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}|\mathbf{A}_{11}| - a_{12}|\mathbf{A}_{12}| + a_{13}|\mathbf{A}_{13}| = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix}$$

$$= a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31}$$

Cross Product Using Determinant

- The cross product of two 3D vectors can be calculated using determinant as follows:

$$\mathbf{v}_1 \times \mathbf{v}_2 = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix} = \mathbf{i}\begin{vmatrix} y_1 & z_1 \\ y_2 & z_2 \end{vmatrix} - \mathbf{j}\begin{vmatrix} x_1 & z_1 \\ x_2 & z_2 \end{vmatrix} + \mathbf{k}\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$$

$$= (y_1 z_2 - y_2 z_1)\mathbf{i} + (z_1 x_2 - z_2 x_1)\mathbf{j} + (x_1 y_2 - x_2 y_1)\mathbf{k}$$

Summary

- What are the characteristics of a vector?
- What operations are defined for vectors.
- How to calculate the vector projection onto another vector?
- How to calculate cross product? What is the geometric meaning of cross product?
- How to do matrix operations?
- What is a orthogonal matrix?
- How to calculate determinant?