1. Introduction

R. Mukundan,
Department of Computer Science and Software Engineering
University of Canterbury.

1.1 Computer Graphics

Computer graphics is a rapidly expanding field of computer science, with applications spanning a wide range of areas such as game engine development, mobile graphics technology, cartoon animations, computer-generated commercials, algorithm and information visualization, medical data analysis, process simulation, image processing, user interface design, augmented reality and movie special effects. Until a few decades ago, development of graphics applications required expensive hardware and display systems, and substantial amount of add-on computer resources. Graphics applications have now become ubiquitous, and we encounter them every day on television, laptops and mobile devices. Graphics algorithms are being increasingly used in many scientific and technological areas, with an explosive growth in applications requiring three dimensional scenes, models and animations. On present-day processors, we are able to render highly complex three-dimensional models and scenes with remarkable photorealistic effects and real-time animation rates. The expansion of computer graphics into diverse and interdisciplinary areas is the result of many factors such as decreasing hardware costs, availability of powerful graphics processors and associated software tools, research advancements in the field, and significant improvements in graphics APIs. Additionally, vast amounts of resources including online learning materials, images, 3D models, libraries, programs, data sets and research papers are now easily available to developers of computer graphics applications.

It should be noted here that “computer graphics” and “graphics design” often used as synonymous terms, denote two entirely different fields. While graphics design refers to creative processes (possibly using software packages such as Photoshop) for visual communication, computer graphics can be formally defined as the field of study of methods, algorithms, mathematical techniques and software implementations for digitally synthesizing and manipulating visual content.

1.2 Graphics Application Development

Modeling, rendering and animation are typically the main processes involved in the development of a graphics application (Fig 1).

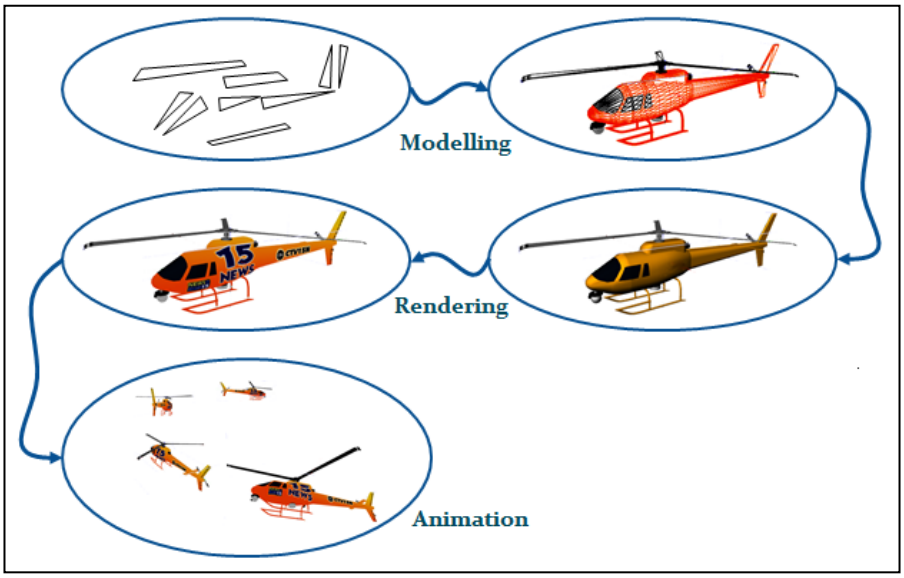
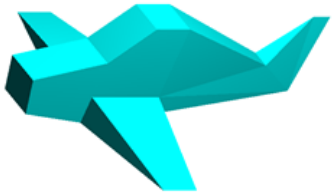


Fig. 1. Processes in the development of a graphics application.

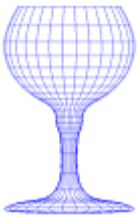
Modelling: Three-dimensional modeling refers to the process of creating scene content using one or many object models. An object might have a simple geometry containing only a few polygons (eg., a cube) or could represent a highly complex shape (eg. a terrain) consisting of several thousand polygons. An object model is made up of *graphics primitives*, usually triangles or quadrilaterals. A modeling process normally involves several transformations (scaling, rotations and translations of polygons). A model can be created using various ways:



A set of graphics primitives can be selected and carefully positioned and orientated to construct a shape. Simple shapes can be created in this fashion. You may require a graphical sketch of the model for estimating the positions of the polygonal elements.

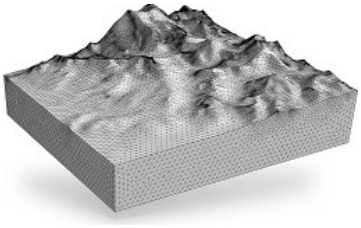
Graphics libraries generally support a few *built-in models* such as the cube, octahedron, sphere, torus, and the most popular teapot. We can create *composite models* by combining a set of built-in models using appropriate transformations. For example, a simple human character model may be created using scaled, rotated and translated versions of a few cubes.

We can procedurally create models using the mathematical equations that represent a shape. Shapes such as spheres, ellipsoids, cones etc., have well-defined mathematical equations that can be used to compute the vertex positions of polygonal surface elements.



We can use surface generation methods such as spline approximations, surface extrusions, surface revolutions and fractal algorithms to create complex shapes using iterative procedures. For example, the revolution of a two-dimensional curve about an axis produces a three-dimensional surface.

There are many types of 3D modelling software which content developers use for creating extremely complex and highly detailed models. Blender, 3DS Max, SketchUp, Maya, MilkShape-3D, ZBrush are examples of such tools.



One could acquire 3D data with the help of laser scanners, range sensors or other distance measurement devices. Modeling algorithms then convert the *point cloud* to polygonal models. Terrain models are often constructed in this manner, and stored as digital elevation maps (DEMs).

Rendering: Rendering refers to the process of displaying a model (or models) by applying colours to polygonal faces and using illumination models to generate lighting effects. Additionally, textures may be mapped to the polygons to generate a photorealistic rendering of objects. Illumination models create variation of shades across a surface, providing shape cues. Shadow generation algorithms are also commonly used to further improve the quality of rendering.

Animation: A series of transformations applied to objects to create motion sequences is collectively called animation. Animations can be defined using functions that modify transformation parameters over time, or with the help of *keyframes* that specify object configurations only at certain points in time. Interpolation between keyframes produces a seemingly continuous motion of objects (or object parts). Animation sequences may also be generated through interactive inputs (mouse motion, keyboard inputs etc.) or through some procedural methods (eg., game logic, simulation outputs).

1.3 Graphics API

In the 1970s and the 80s, only very few graphics libraries were available to developers. Popular among them were the Graphical Kernel System (GKS) and the Programmable Hierarchical Interactive Graphics System (PHIGS). Some of the processors and display systems (workstations) used in those days were Silicon Graphics (SGI), Tektronix (TEK), and Hewlett Packard (HP).

The Open Graphics Library (OpenGL) was released in 1992. It was developed by Silicon Graphics (SGI) as a platform independent, low-level programmer interface to the graphics hardware. OpenGL is often visualized as a state machine, where the operations performed on the rendering pipeline are based on the current settings of the

state machine. OpenGL has now become an industry-standard API for programming 3D graphics and supports a huge range of graphics hardware and features. The current version of OpenGL also includes a powerful shader language designed for *shapder programs* that run on the graphics processor. A chart showing important stages in the evolution of graphics APIs and their current versions is given in slide [1]-3.

The main competitor for OpenGL has been Direct3D developed by Microsoft as a subsystem component of DirectX for use on Windows machines and the .NET framework. The development of Direct3D saw several version upgrades with changes in Windows operating system. The current version of Direct3D (DirectX-12) has a rich feature set and is supported on Windows-10 machines and Xbox-One game consoles.

WebGL was developed in 2009 as cross-platform 3D graphics API for the web. It uses a set of JavaScript programming interfaces, HTML5 <canvas> elements and Document Object Model (DOM) interfaces to create a visual layer for 3D applications on web browsers. The hardware-accelerated 3D functions of WebGL are based on the OpenGL-ES API.

COSC363 aims to provide an introduction to WebGL 2 based on the OpenGL ES 3.0 feature set. WebGL-2 based implementations extensively use *shader programs* and therefore require a good understanding of OpenGL-4 API.

1.4 Glut Built-in Objects

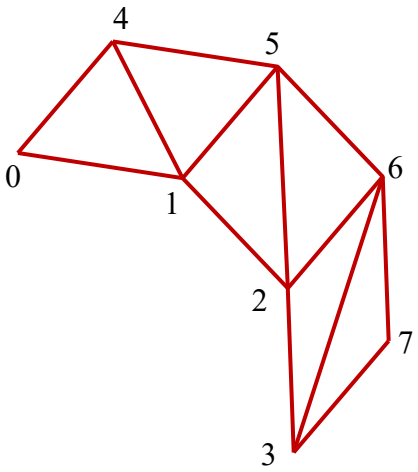
The GLUT (or freeglut) library contains a set of built-in models that can be easily added to a three-dimensional scene. The objects are always created at the origin of the coordinate reference frame. Each object has a set of parameters defining its shape and the number of subdivisions it should have. For a description of the functions and their parameters, please refer to the file GLUT-GLU-Objects.pdf (in Lab01).

There are a few important geometrical properties of GLUT objects that are worth noting here. The cone does not include its base, and is actually a hollow cone (even though its name is "glutSolidCone"!). The famous teapot is probably the best example of a surface generated using Bezier patches. We will learn about Bezier surfaces and the construction of the teapot itself in the second half of the course. The five solids Tetrahedron, Cube, Octahedron, Dodecahedron and Icosahedron are called *platonic solids*. These are regular polyhedrons which form approximations of the sphere consisting of equilateral and equiangular faces. Only 5 platonic solids exist. The cube has 8 vertices and 6 faces, while the octahedron has 6 vertices and 8 faces. The cube and the octahedron are therefore called *dual surfaces*. Similarly, the dodecahedron (20 vertices, 12 faces) is the dual of the icosahedron (12 vertices, 20 faces).

The GLU library contains two useful object models: the cylinder and the disk. The cylinder model has a general representation that allows different radii to be defined for the base and the top. Thus we can use the model to create tapered cylinders or cones. The disk model is often used as the base for cylinders and cones.

1.5 3D Mesh Objects

The geometrical structure of a mesh object can be conveniently stored in two tables: the vertex table and the polygon table. The vertex table is a simple list of vertices with the positional information of each vertex stored as x, y, z coordinate values. The polygon table gives the definitions of each polygon in terms of vertex indices. A vertex index is an integer value that is used as the array index for retrieving the coordinates from the vertex table. After vertex coordinates and polygon definitions, the next most important information associated with mesh data is the normal definition. Normal vectors are required for lighting computation. They may be attached to polygonal faces or vertices. The normal vectors for each triangle can be computed inside the rendering application using the data stored in vertex and polygon tables. However, some mesh files contain a separate "normal table" of pre-computed normal vector components which can simply be read-in by an OpenGL application.



Vertex Table

	x	y	z
0	0.0	1.75	0.5
1	1.0	1.75	0.5
2	1.6	1.0	0.5
3	1.6	0.0	0.5
4	0.0	1.75	−0.5
5	1.0	1.75	−0.5
6	1.6	1.0	−0.5
7	1.6	0.0	−0.5

Polygon Table

i	j	k
0	1	4
4	1	5
1	2	5
5	2	6
2	3	6
6	3	7

An example showing a simple mesh segment and the corresponding mesh data.

Complex mesh models with very large polygon counts are usually stored in compressed and binary formats to save file space. Simple mesh geometries on the other hand can be stored in ASCII formats, where the files can be easily viewed using commonly available text editors. Some of the popular ASCII file formats are OBJ, OFF and PLY. The Object (.OBJ) format was developed by Wavefront technologies. This format allows the definition of vertices in terms of either three-dimensional Cartesian coordinates or four-dimensional homogeneous coordinates. Polygons can have more than three vertices. In addition to a basic set of commands supporting simple polygonal mesh data, the .OBJ format also supports a number of advanced features such as grouping of polygons, material definitions and the specification of free-form surface geometries including curves and surfaces.

The Object File Format (.OFF) is another convenient ASCII format for storing 3D model definitions. It uses simple vertex-list and face-list structures for specifying a polygonal model. Unlike the .OBJ format, this format does not intersperse commands with values on every line, and therefore can be easily parsed to extract vertex coordinates and polygon indices. This format also allows users to specify vertices in homogeneous coordinates, faces with more than three vertex indices, and optional colour values for every vertex or face.

The Polygon File Format (.PLY) also organises mesh data as a vertex list and a face list with several optional elements. The format is also called the Stanford Triangle Format. Elements can be assigned a type (int, float, double, uint etc.). This file format supports several types of elements and data, and the complete specification is included in the file header. Parsing a PLY file is considerably complex than parsing an OBJ or OFF file. The PLY format was developed in the 90s by the Stanford Graphics Lab for storing models created using 3D scanners. One such popular model is the "Stanford Bunny" consisting of about 69000 triangles.