Lecture2 Programming Computer Graphics and Visualization

1. Brief history of computer graphics (Not Examinable)

1950: Different software tools from different labs and vendors. Each graphics device had to be programmed individually

1970: Graphics software standardization (GKS, CORE, PHIGS): common language to understand each other, device independence paradigm.

1900: Strong MS Windows SDK influence (Win 3.1). Evolving OpenGL, VRML, graphics formats

Now: Well established and commonly used multi-platform and device independent software tools which are available as international standards and de-facto standards. Standard graphics formats for data exchange

2. Software classification

Classifications

Programming paradigms

- **Imperative style**: tells the computer how to do things step by step
    - eg. C move variable, shift variable
- **Declarative style**: tells the computer what to do

Software libraries & Specially developed graphics languages/systems

Extensions of programming languages

- Math libraries
- Graphics libraries

Other special software is to solve some specific graphics problems

Computer graphics problems

To achieve as fast as possible point-, line- and polygon-based rendering and with as many as possible elements of rendering

Data visualization problems

To represent graphically various data which may have no obvious graphical appearance at all

3. Computer graphics software for visualization

- Imperative style (polygon-based)
  - **OpenGL**, WebGL, Java3D, DirectX
- Declarative style, ray-tracing (pixel precision)
  - **POV-Ray**
- Declarative style (polygon-based)
  - **VRML, X3D**

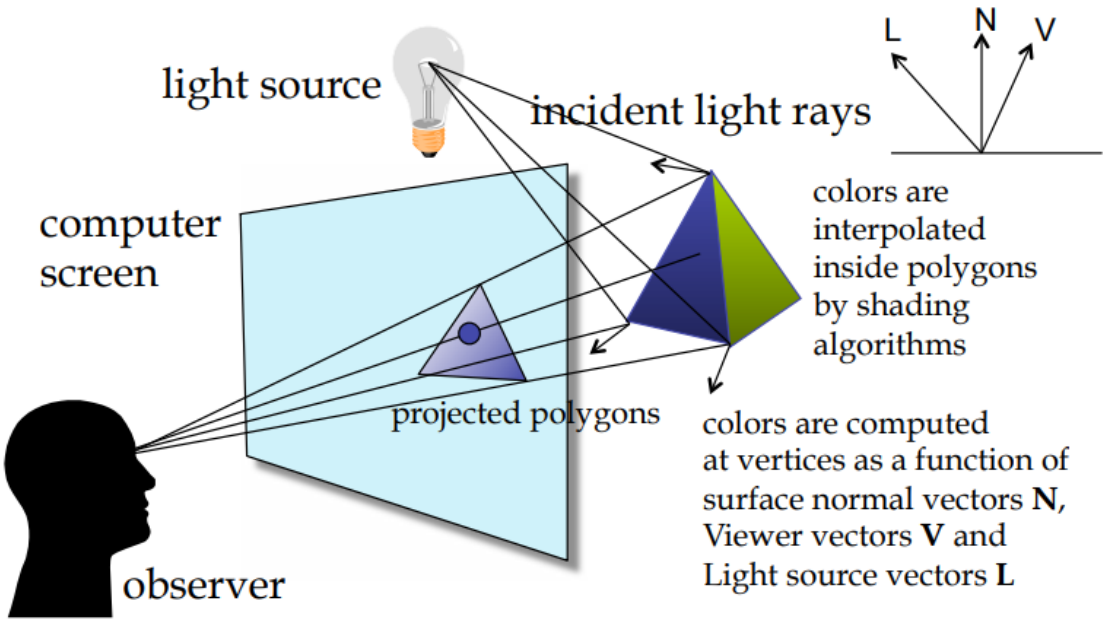
OpenGL

> http://www.opengl.org/

- De facto standard graphics software
- Part of MS Visual Studio, Apple Xcode, etc. C**ross-language, cross-platforms API**
- **Multi-platform software interface** to graphics hardware
- About 120 distinct commands which can be used to specify objects and operations needed to produce interactive three-dimensional applications
- Allows for **defining 2D/3D points, lines, and polygons**. It supports **affine (translation, rotation, scaling), orthographic and perspective projection transformations**. Other features are **RGBA and color index** display modes, multiple **light sources**, **blending**, **antialiasing**, **fog**, **bitmap** operations, **texture mapping** and **multiple frame-buffers**
- Chapter 7 “Let’s Draw”, Alexei Sourin. Making Images with Mathematics, Springer, 2021.

Polygon-based Visualization

> http://www.povray.org/



- Define objects eventually use **polygons(triangles)**
- Those projects are **projected onto an image plane** to create images

- Set up a **light source**, and the distance of observer to the plane, etc.
- Colors are **interpolated** inside polygons. In the vertex, those colors are computed by some formulas of **surface normal vectors, viewer vectors and light source vectors**

OpenGL Example

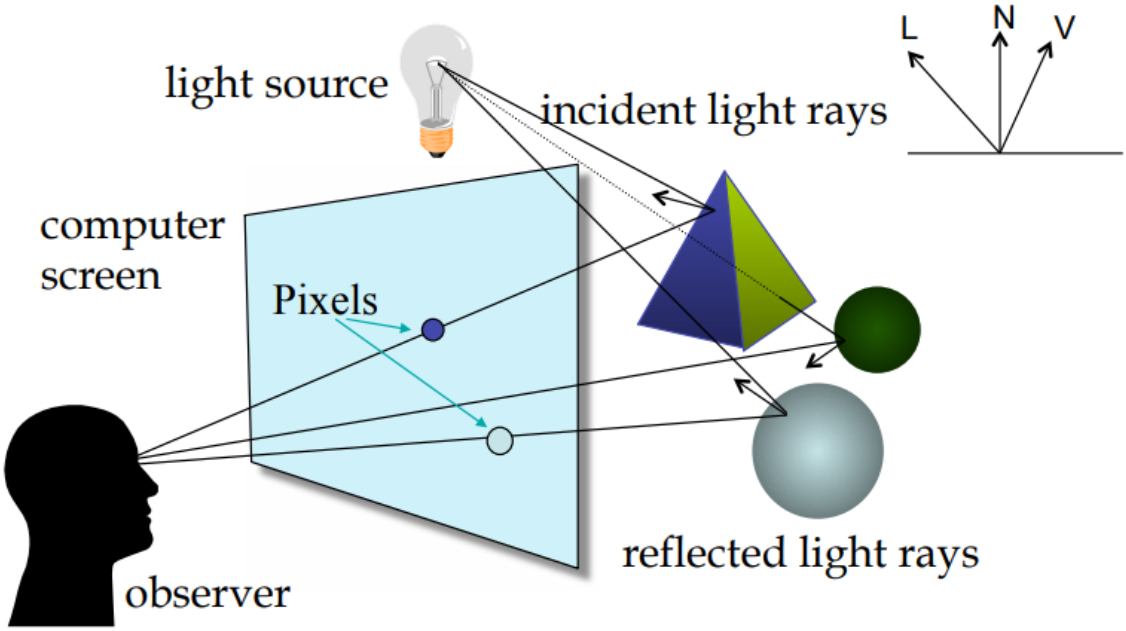
```
main()
{
    Open_A_Window(); // to be implemented by the user
    glClearColor(0.0, 0.0, 0.0, 0.0); // clear the color with the black color
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0); // set the foreground color to be white
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    glBegin(GL_POLYGON); // define the white squares
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush(); // flush the computer to display graphic in the computer
    Keep_The_Window_On(); // to be implemented by the user
}
```

POV-Ray

> http://www.povray.org/

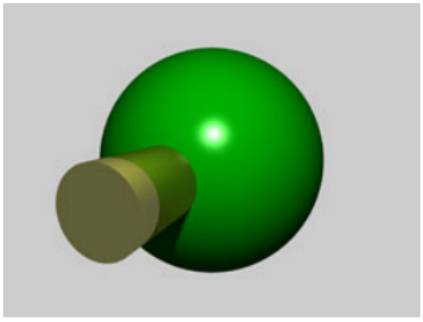
- **POV-Ray (Persistence of Vision Ray Tracing)** is a copyrighted freeware program that lets a user easily create fantastic, three-dimensional, photorealistic images on just about any computer
- Scene definition language describes shapes, colors, textures and lighting in a scene
- A large set of **3D shapes**, affine transformations, **Boolean operations**, multiple light sources, ability to render shadows, rendering with antialiasing, and different photorealistic techniques including textures
- Mathematically simulates the rays of light moving through the scene to produce a photorealistic image
- De facto standard for ray tracing
- Chapter 7 “Let’s Draw”, Alexei Sourin. Making Images with Mathematics, Springer, 2021.

Pixel-precision(Ray-Tracing) Visualization



- From the eye of the observer, generate rays which go back to the scene through the image plane
- Image plane is created by pixels
- Cast rays through every pixel
- Check whether the vector ray will intersect any of the objects
- If intersects, we have to compute a reflect ray, and also the normal vectors, light vectors etc.

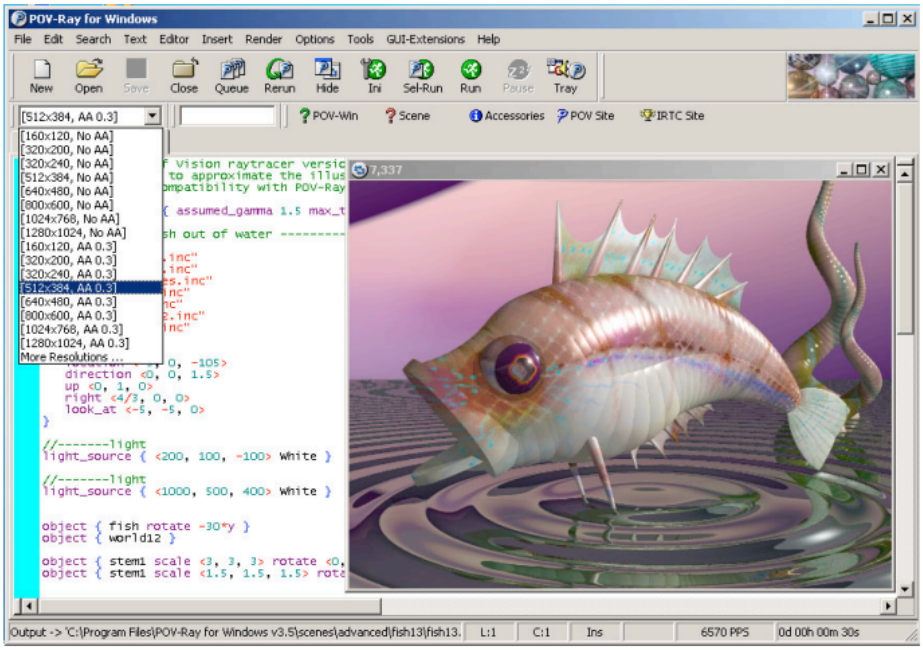
POV-Ray Example



```
camera {
    location <2, 2, -3.5>
    up <0, 1, 0>
    right <4/3, 0, 0>
    look_at <0, 1, 2>}
    light_source { <2, 4, -3>
    color White
}
union{
    sphere {<0, 1, 2>, 2
    texture { pigment {color Green} finish {phong 1} } }
    cylinder {<0,1,-1.2>,<0,1,4.2>, 0.5
    texture{Gold_Metal} }
```

```
  }
background { color rgb <0.8, 0.8, 0.8> }

```



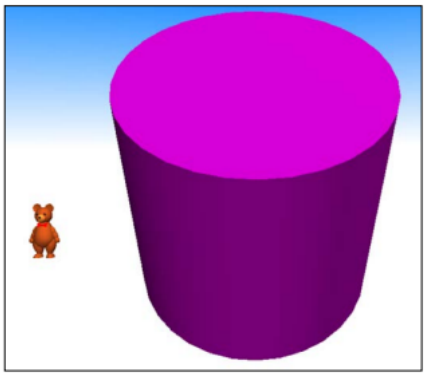
VRML / X3D

> http://www.web3d.org/

- **Virtual Reality Modeling Language** (VRML) and Extensible 3D (X3D)
- Royalty-free open ISO standards file format for describing interactive 3D objects and worlds. Designed to be used on the Internet, intranets, and local client systems. Intended to be a universal interchange format for integrated 3D graphics and multimedia. May be used in a variety of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages, and shared virtual worlds.

- Capable of representing static and animated dynamic 3D and multimedia objects with hyperlinks to other media such as text, sounds, movies, and images. VRML/X3D browsers, as well as authoring tools for the creation of VRML files, are available for many different platforms.
- Chapter 7 “Let’s Draw”, Alexei Sourin. Making Images with Mathematics, Springer, 2021.

VRML/X3D Example



```
#VRML V2.0 utf8
Transform { translation 5 0 0 rotation 1 0 0 0.7  // define the shape
    Children[ Shape {
    appearance Appearance {material Material {
    diffuseColor .5 0 .5 shininess .5 } }
    geometry Cylinder {radius 3 height 6
    side TRUE top TRUE bottom TRUE } } ] }

Inline {url "http://…/bearav.wrl"} // hyperlink to another code

PointLight {on TRUE ambientIntensity 1
color 1 1 1 location 250 400 150 radius 1500 }

Background {
 skyColor [0 0 1 0 .5 1 1 1 1 ] skyAngle [1.309 1.571] }
```

Python

- Computer graphics is computationally heavy and needs high performance
- Python is for scripts with core engines made in C, C++
- Basic computer graphics can be made, for example, with graphics.py, or using Python binding for
    - OpenGL: PyOpenGL http://pyopengl.sourceforge.net
    - Python Computer Graphics Kit http://cgkit.sourceforge.net,
    - Panda 3D https://www.panda3d.org can be used for making games but internally it uses C++

PyOpenGL

PyOpenGL is interoperable with a large number of external GUI libraries for Python including(but not limited to):

- wxPython

- PyGame
- PyQt and PySide
- PyGTK
- Raw XLib
- OSMesa
- Raspberry Pi BCM
- Tkinter

PyOpenGL 3.x runs on:

- Python 3.3+ (3.2 support is likely to work, but untested)
- Python 2.7(recommended)
- Python 2.6 (for compatibility with older software and systems)
- PyPy (experimental)

4. Summary

- **Polygon-based visualization** is **fast** but it **compromises on precision of presentation**. Geometry of fine details is often replaced by image **textures** (patterns) displayed on the surfaces
- **Ray-tracing** is very **precise** but **slow**. Mostly used for making images and not designed to be an interactive visualization too
- VRML/X3D is **polygon-based**, **declarative style virtual scene description language**. Compared to OpenGL and other graphics libraries, it requires very little time to start programming both web-enabled and local visualization problems. It is full of technological solutions but may require a deeper learning to master them.