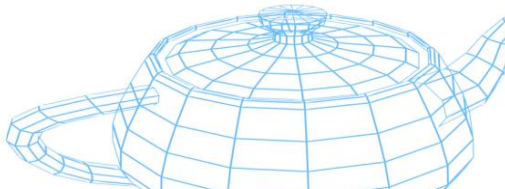
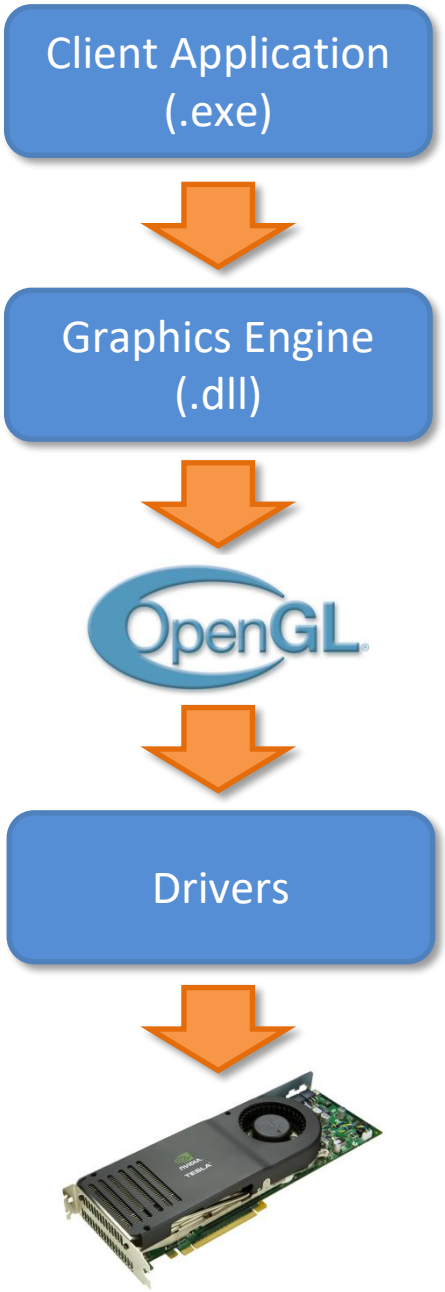
SUPSI

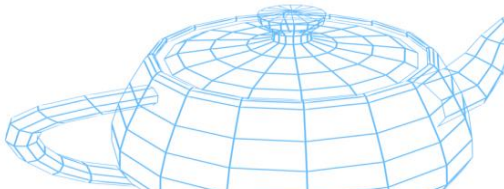
Computer Graphics

OpenGL (1): overview and first steps

Achille Peternier, adjunct professor




Client Application
(.exe)
Graphics Engine
(.dll)
OpenGL
Drivers



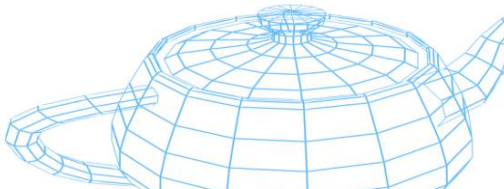
OpenGL overview

- **Open G**raphics **L**ibrary.
- Cross-platform and -language API for 2D/3D computer graphics.
- Abstract API:
  - Can be implemented both in hardware and software.
- Maintained by the non-profit group Khronos.

OpenGL overview

- Rendering only:
  - No audio.
  - No image file processing.
  - No user input.
  - No GUI.
  - No timing.
  - No animations.
  - No multithreading.

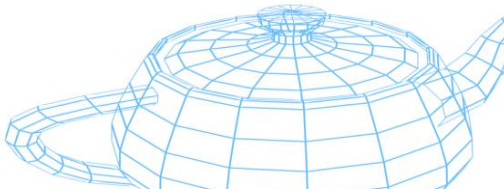
Just raw primitive drawing!



OpenGL overview

- Open specification:
  - Not *open-source*: there's nothing to download (except the document defining the specs).
  - Opposed to ISO standards, where you must pay for using them.
  - You can read the specs and make your device accessible through your own implementation of the specs:
    - Mesa 3D is an open-source *implementation* of the open *specification.*

- The specification is maintained and (unfortunately no longer…) regularly updated by the Khronos Group.

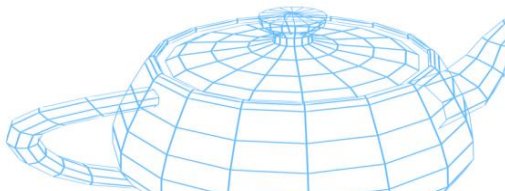


Using OpenGL

- Include **`GL/gl.h`**

- On Windows:
  - Link your code to **opengl32.lib** (also when building for x64):
    - **`opengl32.dll`** loads and checks for a real OpenGL driver:
      - The driver is typically installed when you install your full graphic drivers (and not with the default drivers installed by Windows).
    - **`opengl32.dll`** belongs to Microsoft and features OpenGL 1.1:
      - For more recent versions, use extensions or (much better) an external library that does the job for you (e.g.: Glee, GLEW).

- On Linux:
  - Link your code to **libGL.so**



Using OpenGL

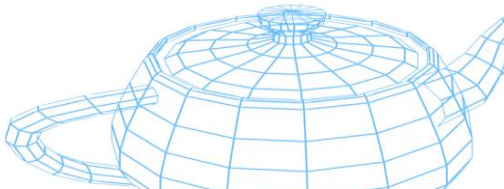
- Some notebooks have more than one graphics card:
  - Integrated GPU (e.g., Intel HD graphics);
    - Inferior 3D performance, but longer battery life.
  - Discrete GPU:
    - Ideal for 3D gaming and CAD, but more power-hungry.
  - Nvidia Optimus and AMD Dynamic Switchable Graphics use this technology: make sure that you run your OpenGL application with the correct profile.

- On Windows, without a proper driver installed, a software emulation of OpenGL 1.1 is provided (reported as “Windows GDI”).

- On Linux, without proprietary drivers, a software, open-source implementation of OpenGL (up to recent versions, depending on the available hardware) is provided (reported as “Mesa”).

Using OpenGL

- Requires the initialization of a **context**:
  - A context is a container allocated by the driver to store the state of the API and the graphic resources used by a windowed application.
  - Context creation is platform-specific.
  - This part is not portable:
    - Software needs one specific initialization procedure for each platform.
    - Auxiliary libraries are often used to deal with this aspect.

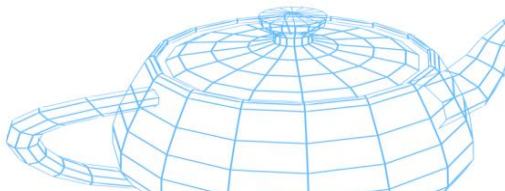


Using OpenGL

- Once the context is initialized, use **`glGetString()`** to get NULL terminating strings with several pieces of information, like:
  - **`GL_VERSION`** OpenGL supported version.
  - **`GL_VENDOR`** driver's implementer.
  - **`GL_RENDERER`** renderer used (usually the name of the GPU).
  - **`GL_EXTENSIONS`** a (long) list with all the supported extensions.

OpenGL context information example

```
version  . . :  4.6.0 Compatibility Profile Context .220823
vendor . . . :  ATI Technologies Inc.
renderer . . :  Radeon (TM) Pro WX 9100
```



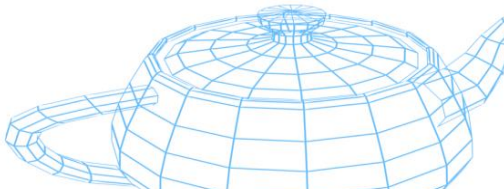
FreeGLUT

- **Free** alternative to the Open**GL U**tility **T**oolkit.

- Evolution/clone of GLUT, originally created by Mark Kilgard (Nvidia) and abandoned in 1999:
  - Less restrictive license.

- FreeGLUT started in 1999 by Pawel Olzsta:
  - Open-source.
  - No code inherited from GLUT.

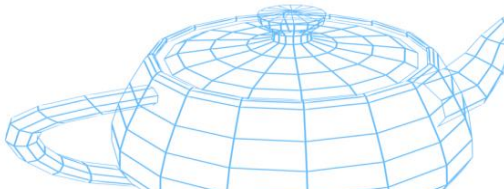
- Current version: **3.4.0** (7 October 2022!)



FreeGLUT

- Feature list:
  - Event-based (callbacks).
  - Supports user-input through mouse and keyboard.
  - Provides a menu/sub-menu system.
  - Supports printing text to the OpenGL window.
  - Provides a series of built-in, dynamically generated 3D objects.
  - Available on Windows, Linux, MacOS, etc.
  - …

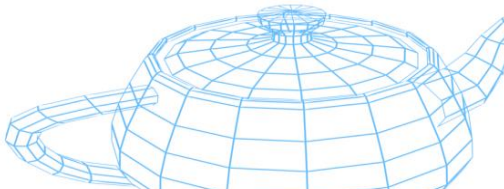
- Typical usage:
  - Initialize the library.
  - Create a window for graphic output.
  - Register callback functions.
  - Enter the main loop.



FreeGLUT

- Windows:
  - Download and compile the project:
    - http://freeglut.sourceforge.net/
  - Use the .lib + .dll or .lib-only (static) version.
  - Define **`FREEGLUT_STATIC`** for static linking:
    - The static lib is included (and used) in the tutorials and series' solutions.

- Ubuntu:
  - Execute: **`sudo apt install freeglut3-dev`**
  - Link to *libglut.so* (dynamic) or *libglut.a* (static).

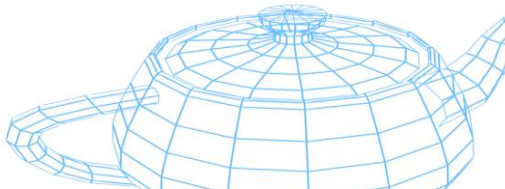


FreeGLUT

- **`void glutInit(int *argc, char *argv[]);`**
  - Accepts -display, -geometry

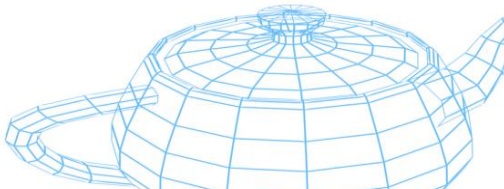
- **`void glutInitDisplayMode(flags);`**
  - Accepts:
    - **`GLUT_SINGLE`**, **`GLUT_DOUBLE`** single/double buffering
    - **`GLUT_RGB`**, **`GLUT_RGBA`** color mode
    - **`GLUT_DEPTH`** enables the z buffer

    ...
  - E.g.: **`glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA);`**



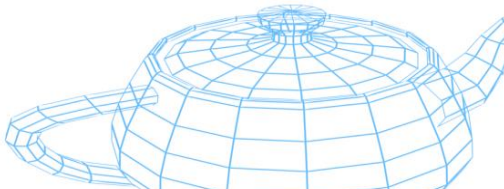
FreeGLUT

- **`void glutInitWindowSize(int width, int height);`**
- **`void glutInitWindowPosition(int x, int y);`**
- **`int glutCreateWindow(char *name);`**
  - name = window title.
  - returns the window ID.
  - The OpenGL context is initialized within this method: you can start using the OpenGL API only **after** its invocation.



FreeGLUT

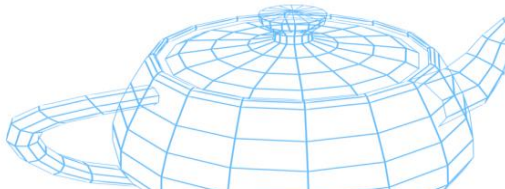
- Callback registration:
  - **`glutDisplayFunc(func. ptr.);`**
    - Invoked each time the output scene must be re-rendered.
    - Call **`glutPostWindowRedisplay(winId)`** to force a refresh.
  - **`glutReshapeFunc(func. ptr.);`**
    - Triggered each time the window size is changed.
  - **`glutMouseFunc(func. ptr.);`**
    - Triggered each time the mouse is used.
  - **`glutKeyboard(func. ptr.);`**
    - Triggered each time a keyboard key is pressed.
  - **`glutSpecial(func. ptr.);`**
    - Same as before, but for special keyboard keys such as the arrows.




Tutorial
Basic FreeGLUT
VIDEO
DISPLAY

Main buffers

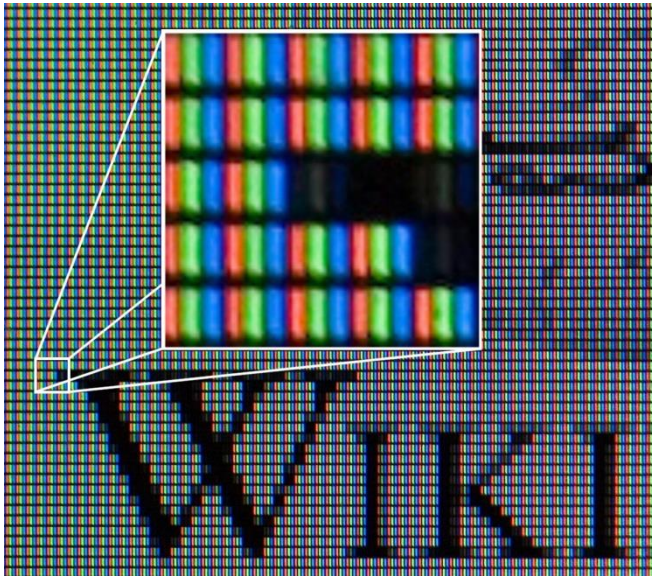
- A rendering-context is initialized by specifying the characteristics of these four buffers:
  - **Framebuffer** (or **color buffer**) = main output buffer (where you render the final image).
  - **Back buffer** = the hidden twin of the framebuffer, where the next frame is being rendered [almost mandatory].
  - **Depth buffer** (or **Z buffer**) = stores Z values for each pixel of the framebuffer [optional].
  - **Stencil buffer** = additional buffer for per-pixel logic operations [really optional].
  - **Accumulation buffer(s)** = one or more additional buffers for storing a series of images [extremely optional, removed from modern OpenGL].

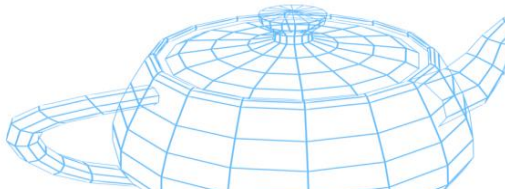


Framebuffer

- Memory segment containing the image information to display through the graphics device.

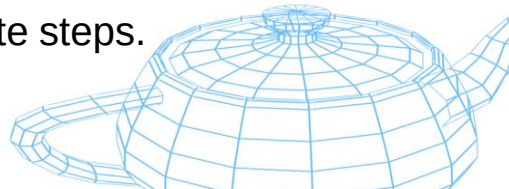
- Contains pixel colors:
  - RGB on modern displays.
  - Single bits for older, monochromatic monitors.





Framebuffer

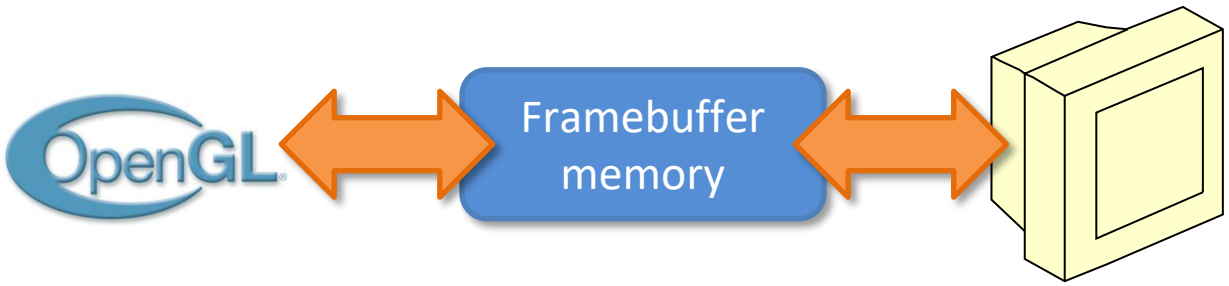
- Different framebuffer color depths:
  - RGB:
    - 1 bit = monochromatic.
    - 4 bit = 16 colors (from a palette).
    - 8 bit = 256 colors (from a palette).
  - RGB/RGBA:
    - 15 bit = high-color (2^{15} colors):
      - 5+5+5 (RGB).
    - 16 bit = high-color (2^{16} colors):
      - 5+5+5+1 (RGBA) = 15 bit high-color + alpha channel.
      - 5+6+5 (RGB) = 16 bit high-color:
        - » Human eye is more sensitive to green.
      - 4+4+4+4 (RGBA).
    - 24 bit = true-color (2^{24} colors, ~16 millions):
      - Human eye can recognize up to 10 million colors.
  - RGBA:
    - 32 bit = true-color (24 bit) + 8 bit alpha channel.
  - > 32 bit:
    - High Dynamic Range (HDR), professional devices, intermediate steps.

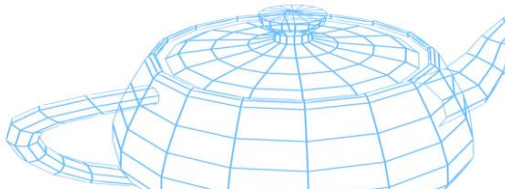


Framebuffer

- The memory stored in the framebuffer is accessed by the device to periodically refresh the pixels rendered on the screen:
  - Typical refresh rate between 60 and 120 Hz:
    - 120 Hz useful for stereographic rendering (60 Hz per eye).

- To avoid visual artifacts (screen flickering and tearing), video memory refreshing is synchronized with the display refresh rate.






Tear Point #1 --->
Tear Point #2 --->

front buffer only

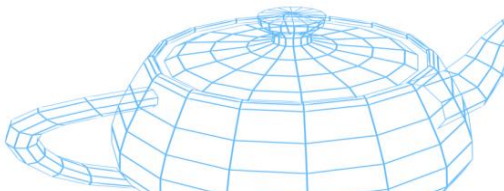
Framebuffer

- Vertical synchronization (a.k.a. v-sync):
  - Old terminology used on CRT monitors:
    - The graphics card waits for the vertical beam to reach the lower-right corner of the screen.
    - While the beam is reset to its upper-left position, the graphics memory is refreshed.
    - Additional modifications are prevented until the next reset.
  - It is used to synchronize the rendering speed with the monitor refresh frequency.

- Double buffering:
  - Uses two buffers: front (main one, rendered on the screen) and back.
  - Instead of rendering to the screen memory buffer, a secondary (hidden) buffer is used.
  - Once the image is ready on the back-buffer, it is copied to the front-buffer (usually during vertical sync):
    - As optimization, back- and front-buffer pointers are swapped (zero copy, page flipping/ping-pong buffering).

back buffer

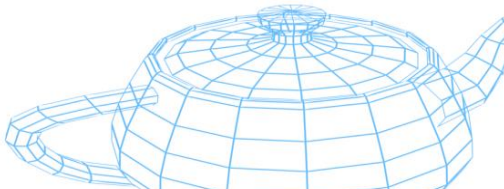
front buffer



Framebuffer

- RGB or RGBA mode must be specified during the context creation:
  - **`glutInitDisplayMode(GLUT_RGB or GLUT_RGBA);`**

- Double buffering must be specified during the context creation:
  - **`glutInitDisplayMode(... | GLUT_DOUBLE);`**
  - Once finished with the rendering of the current frame, front-/back-buffer swapping must be explicitly invoked:
    - **`glutSwapBuffers();`**

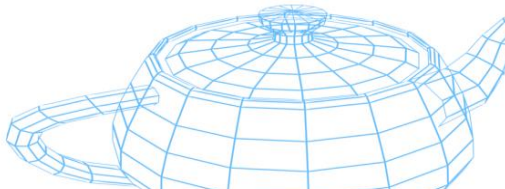


Framebuffer

- Framebuffer clear color is specified through its RGBA components:
  - **`glClearColor(red, green, blue, alpha);`**
    - Alpha is required in RGB mode, too.

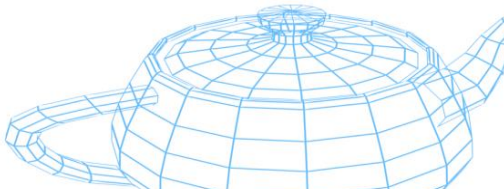
- Framebuffer is cleared explicitly through:
  - **`glClear(GL_COLOR_BUFFER_BIT);`**

- Details about the current context are retrieved through:
  - **`glutGet(enum);`**
    - E.g.: **`GLUT_WINDOW_BUFFER_SIZE`**, **`GLUT_WINDOW_RED_SIZE`**, **`GLUT_WINDOW_GREEN_SIZE`**, **`GLUT_WINDOW_DOUBLEBUFFER`**, …



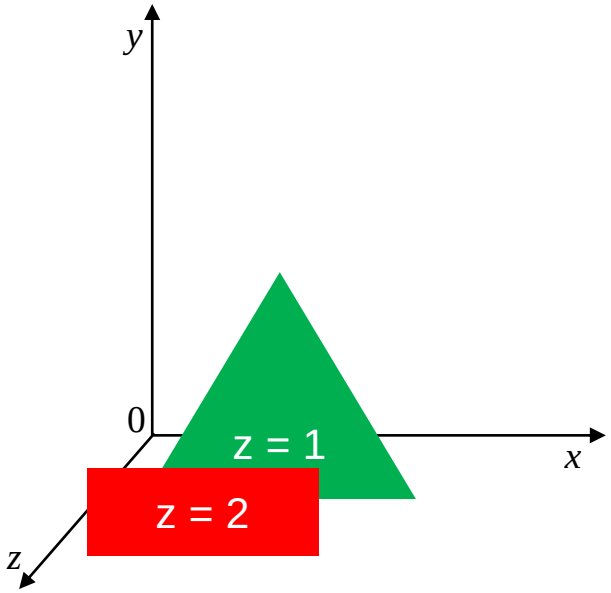
Z buffer

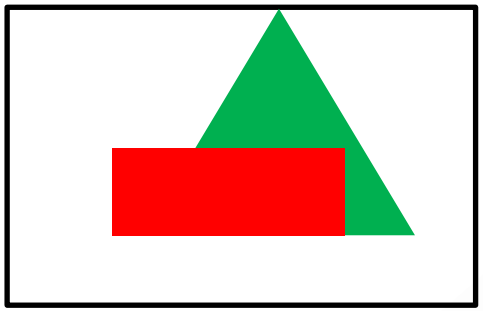
- It is used to store the depth (z) value of each pixel of the framebuffer.
- Works like a “sonar”.
- Operations to the framebuffer are conditioned by the z buffer current state.



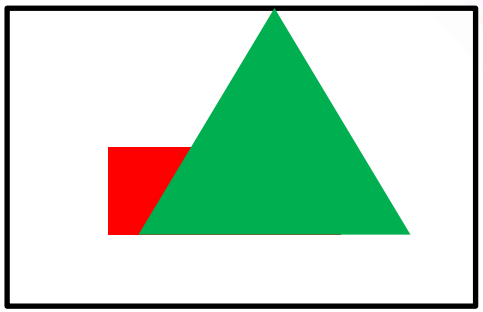
Z buffer

- Without depth test:

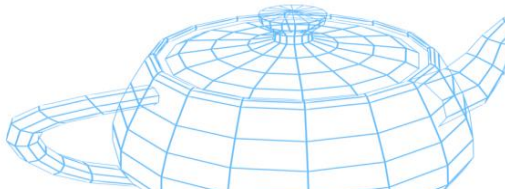




a) first the triangle, then the rectangle

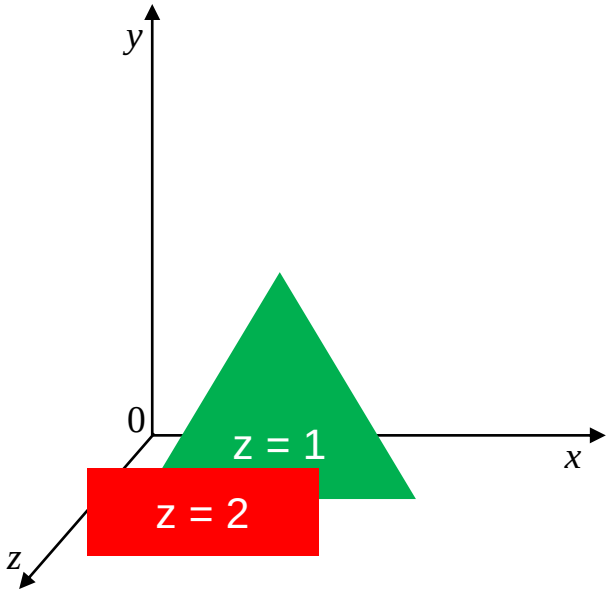


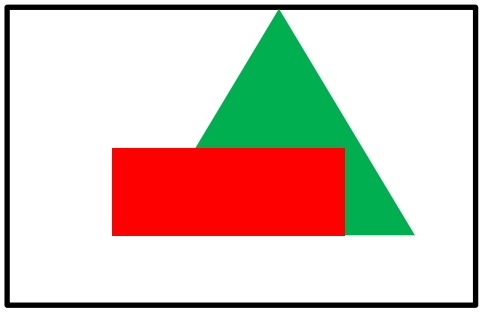
b) first the rectangle, then the triangle



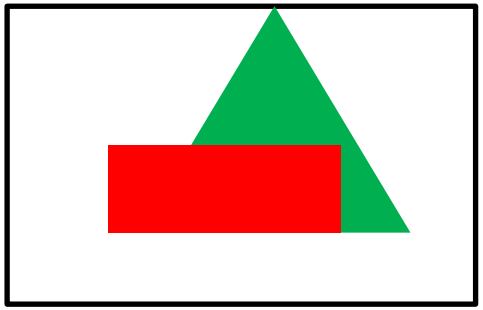
Z buffer

- With depth test: order-independent rendering

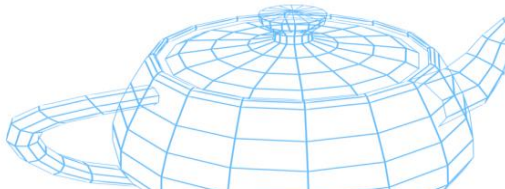




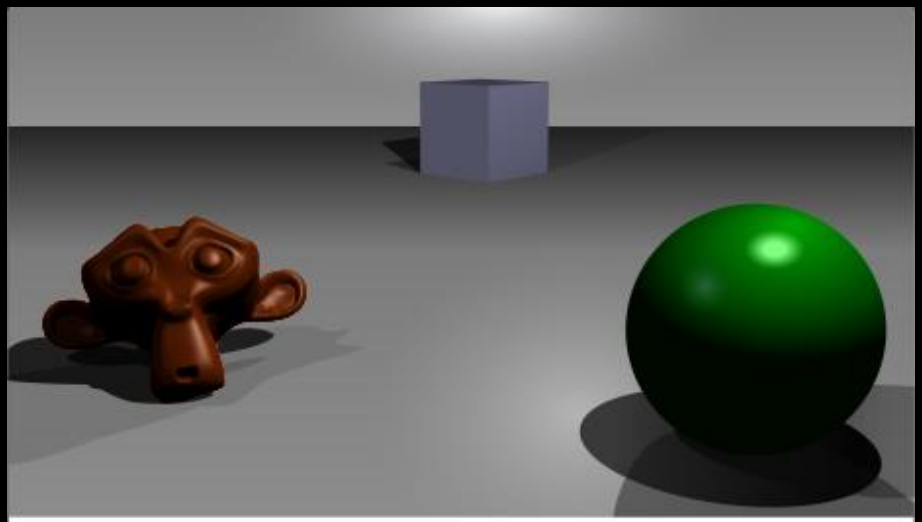
a) first the triangle, then the rectangle



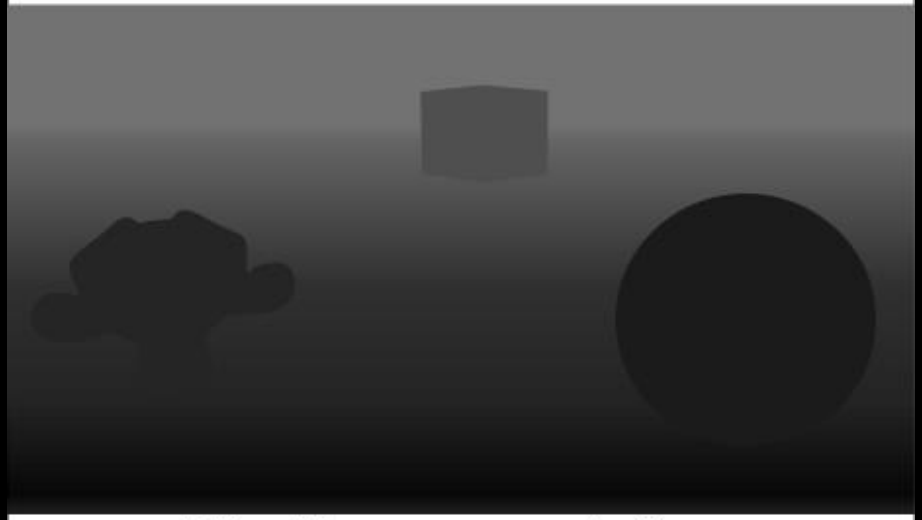
b) first the rectangle, then the triangle



Z buffer



A simple three-dimensional scene



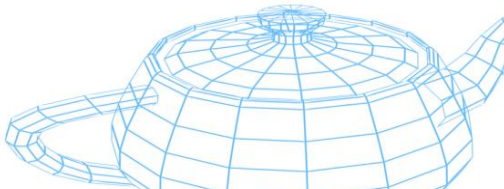
Z-buffer representation

Z buffer

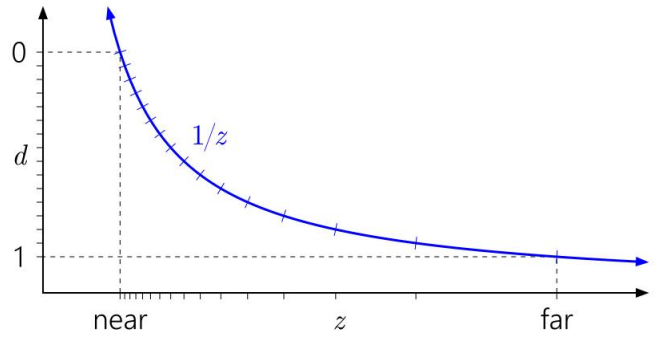
- Contains the perpendicular distance of each pixel of the framebuffer relative to the near clipping plane.

- Values in normalized device coordinates $[-1 < z < 1]$ are resampled into the $[0 < z < 1]$ range.

- Accuracy depends on the used number of bits:
  - Typically 16/24 bit.
  - The 24 bit z buffer is often padded with an 8 bit stencil buffer to reach 32 bit boundaries.
  - 32 bit z buffers are possible only through off-screen framebuffers (via modern OpenGL framebuffer objects).



Z buffer

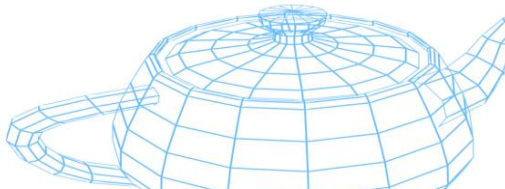


- Precision is higher for objects closer to the *zNear* plane (logarithmic curve):

$$\textbf{Perspective} = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{far + near}{near - far} & \frac{2 \times far \times near}{near - far} \\ 0 & 0 & -1 & 0 \end{bmatrix} \Rightarrow z_{ndc} = \frac{\frac{f+n}{n-f} z_{eye} + \frac{2fn}{n-f}}{-z_{eye}}$$

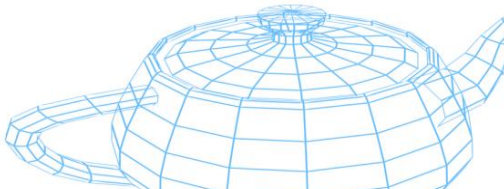
- In general, try to push the *zNear* plane out and the *zFar* plane in as much as possible:
  - Your z buffer accuracy corresponds to the discretization of the space *zFar - zNear* using X bit, where X is your z buffer bit depth.

- **Always** keep *zNear* > 0.



Z buffer

- Must be specified during the context creation:
  - **`glutInitDisplayMode(... | GLUT_DEPTH);`**
- Must be explicitly enabled:
  - **`glEnable(GL_DEPTH_TEST);`**
- Z buffer must be cleaned before rendering:
  - **`glClear(… | GL_DEPTH_BUFFER_BIT);`**
- Z buffer behavior must be configured:
  - **`glClearDepth(float);`** [default is 1.0]
  - **`glDepthFunc(enum);`** [default is **`GL_LESS`**]

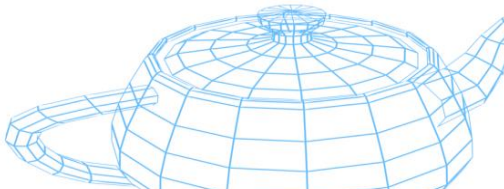


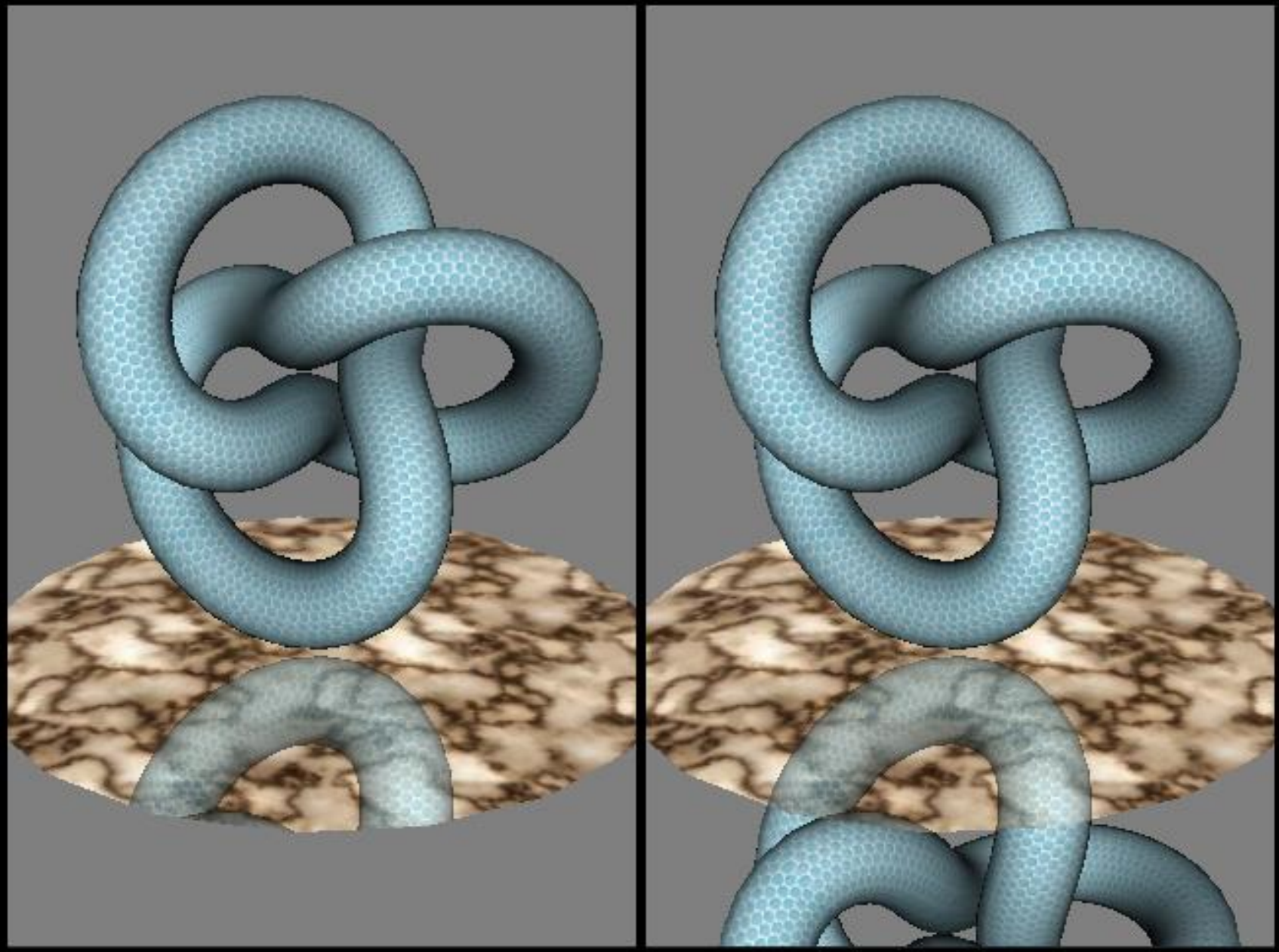
Stencil buffer

- Optional buffer with the same dimension of the color and z buffers and a typical depth of 8 bit:
  - Could be just 1 bit.
  - Interaction with the z buffer:
    - Stencil buffer values can be modified according to the result of the depth test.

- Mainly used to limit the rendering to specific, pixel-precise areas:
  - Planar reflections.

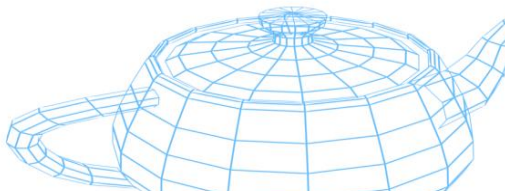
- Other advanced applications involve volume shadows, constructive solid geometry, portals, etc.





Stencil buffer

- Must be specified during the context creation:
  - **`glutInitDisplayMode(... | GLUT_STENCIL);`**
- Must be explicitly enabled:
  - **`glEnable(GL_STENCIL_TEST);`**
- Stencil buffer must be cleaned before rendering:
  - **`glClear(… | GL_STENCIL_BUFFER_BIT);`**
- Stencil buffer's behavior must be configured:
  - **`glClearStencil(int);`** [default is 0]
  - **`glStencilFunc(enum, ref, mask);`**
  - **`glStencilOp(fail, zfail, zpass);`**

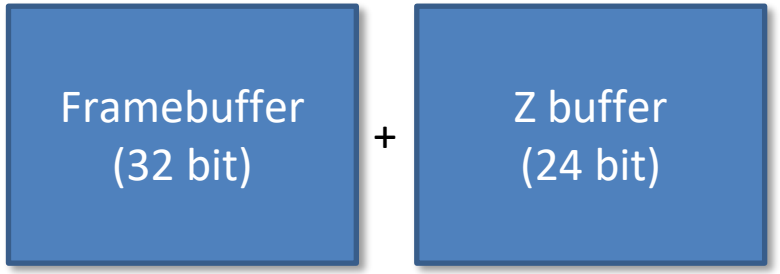


Main buffers

OpenGL context 1 (640x480, no double buffer):

640x480 = 307’200 pixels
24+24+8 = 56 bit
~2 MB of VRAM
(48 MB/s @ 24 fps)

OpenGL context 2 (1920x1080, double buffer):



1920x1080 = 2’073’600 pixels
32+24 = 56 bit
~14 MB of VRAM (~20 MB x2)
(336 MB/s @ 24 fps)