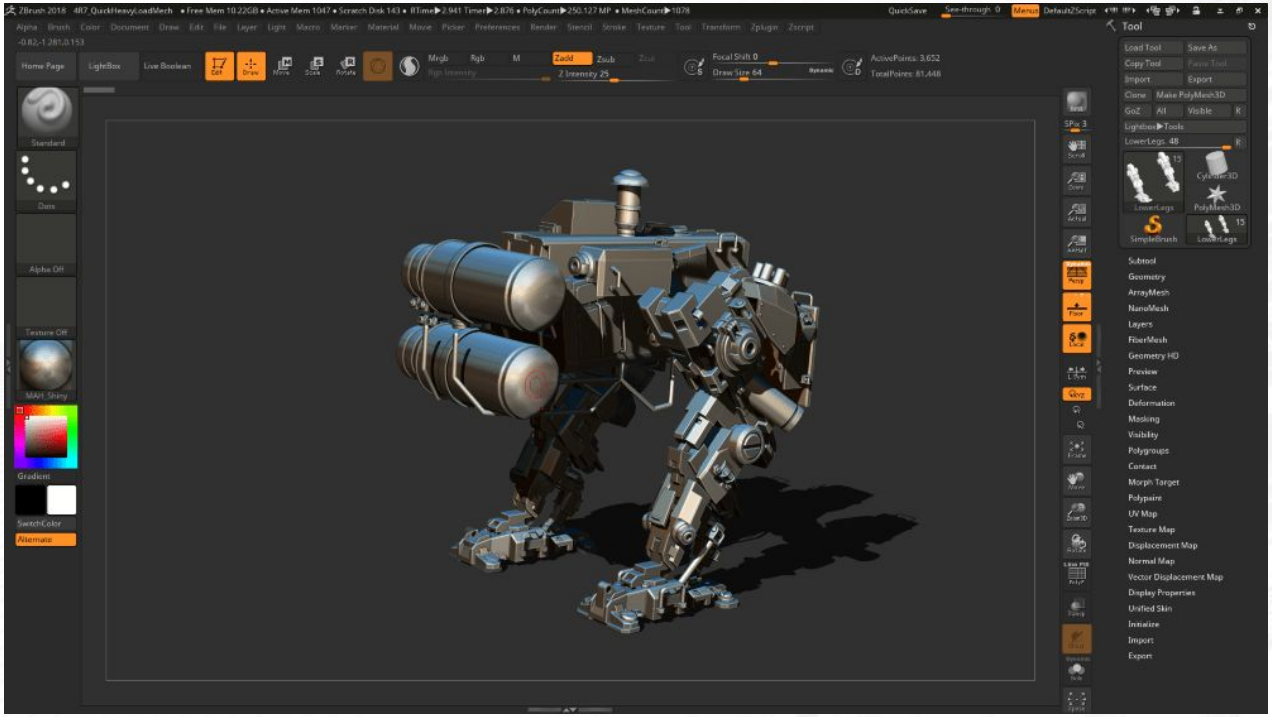
Lesson 6. Fundamentals of 3D Graphics

From a 3D World to a Model



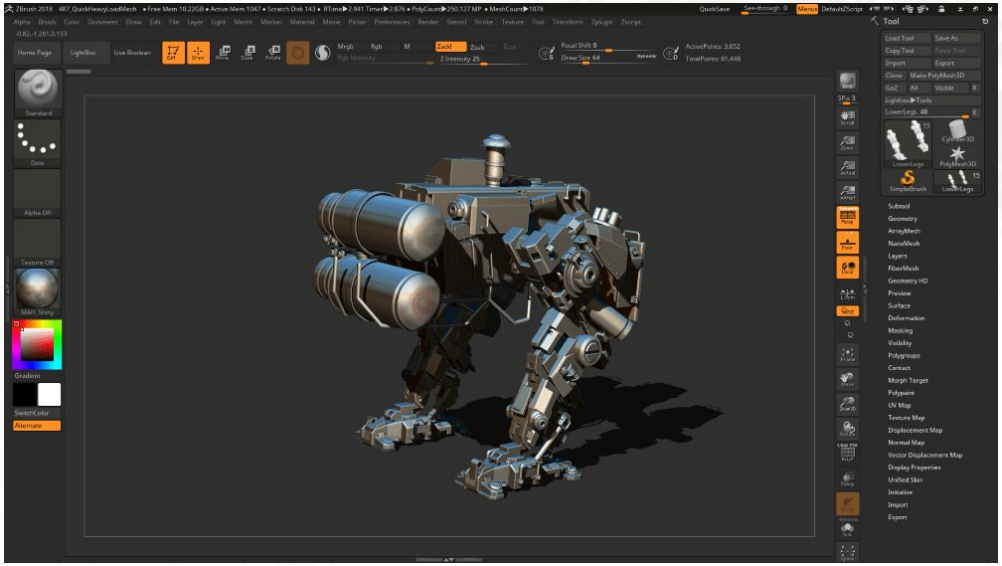
From a 3D World to a Model

Similar to 2D Virtual Worlds:

3D virtual objects and scenarios **need to be defined and stored** in the computer

What Does This Mean?

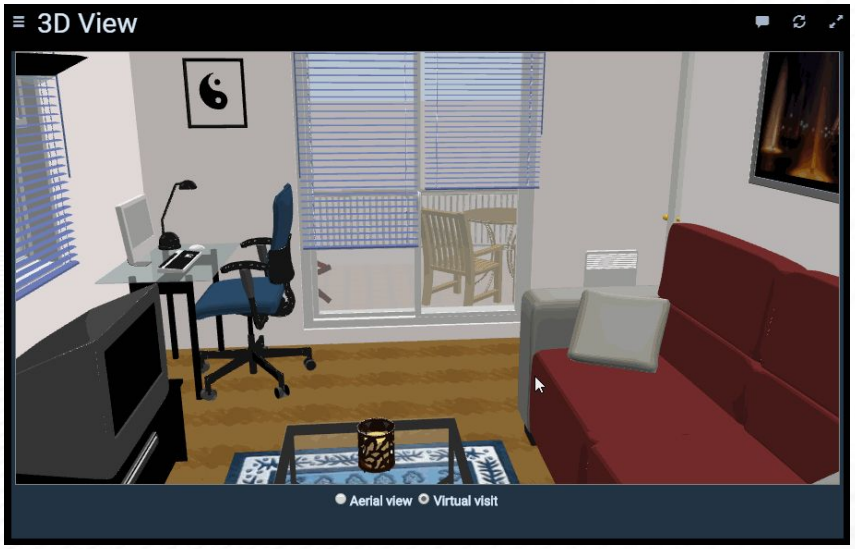
Geometric objects like rectangles, circles, or polygons **must be specified before they are drawn**



How about in 3D?

A three-dimensional world can contain **much more than what is displayed**

From a 3D World to a Model



Despite having more than what is displayed, **the viewer can move around** in the virtual world and explore it

From a 3D World to a Model

Modelling the Object of a Virtual World

Must contain information about **geometry** but also involves other thing such as:

Properties of Surface

Object's Color

Dull or Shiny?

From a 3D World to a Model

Most Objects Do Not Have Existing Counterparts

Usually for **fantasy worlds** in computer games, they do not exist in real life

In these cases, the designer or programmer of the virtual world needs **methods for constructing and modelling 3D virtual objects**

Even if existing objects are to be modelled, construction and modelling techniques are still necessary. (e.g. information that is given is not enough)

From a 3D World to a Model

For Some, Detailed Measurements are Available

3D Laser Scanners provide detailed information about an object's surface geometry. However, they are **usually processed further** with manual corrections.

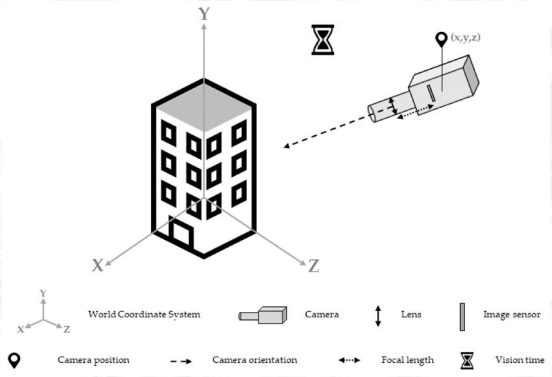
Another example are X-Rays, Ultrasonic and Tomography techniques **provide information about skeletal and tissue structures** from which 3D models of bones and organs can be derived

From a 3D World to a Model

Considerations on Modelling a Virtual World

The first step is to design a virtual world **either manually or automatically from derived measurements**

To represent a part of the virtual world, **the viewer's position and direction of view** in the virtual world must be defined. *Field of View? Viewing Angle? Distance?*

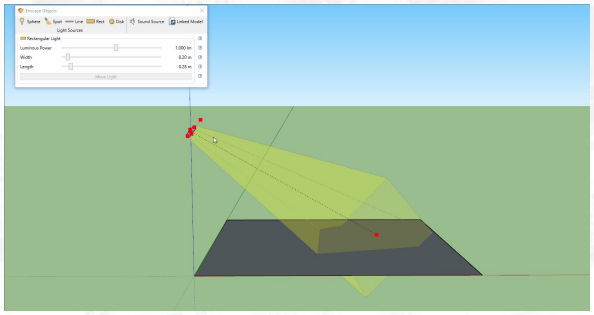


Adding Sound to a Model

Considerations on Modelling a Virtual World

The first step is to design a virtual world **either manually or automatically from derived measurements**

To represent a part of the virtual world, **the viewer's position and direction of view** in the virtual world must be defined. *Field of View? Viewing Angle? Distance?*



The **sources of light** as well as their characteristics must be defined. *Color of the light? Direction?*

From a 3D World to a Model

Considerations on Modelling a Virtual World

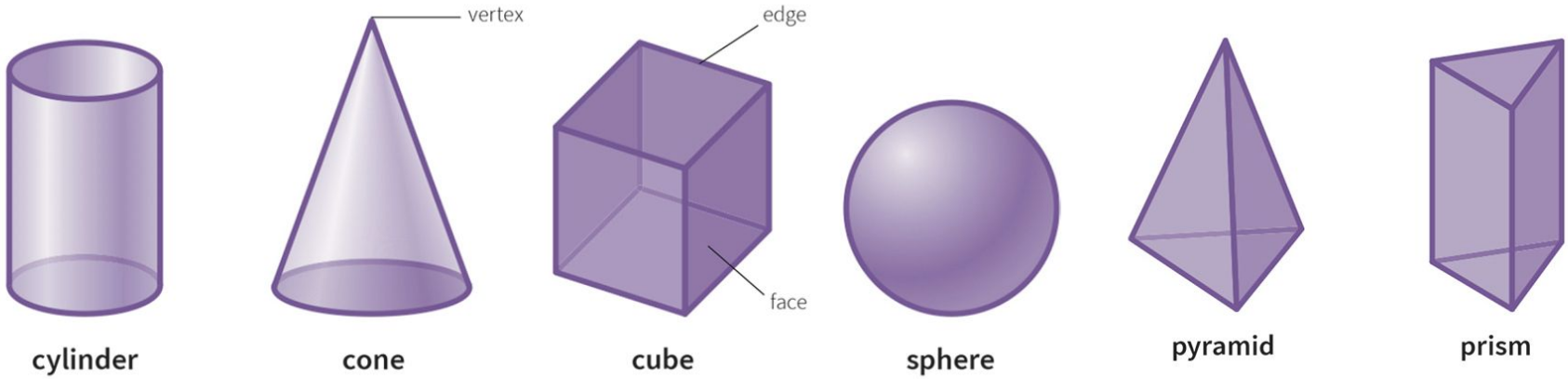
The first step is to design a virtual world **either manually or automatically from derived measurements**

To represent a part of the virtual world, **the viewer's position and direction of view** in the virtual world must be defined. *Field of View? Viewing Angle? Distance?*

The **sources of light** as well as their characteristics must be defined. *Color of the light? Direction?*

Determining which objects are **visible and hidden**

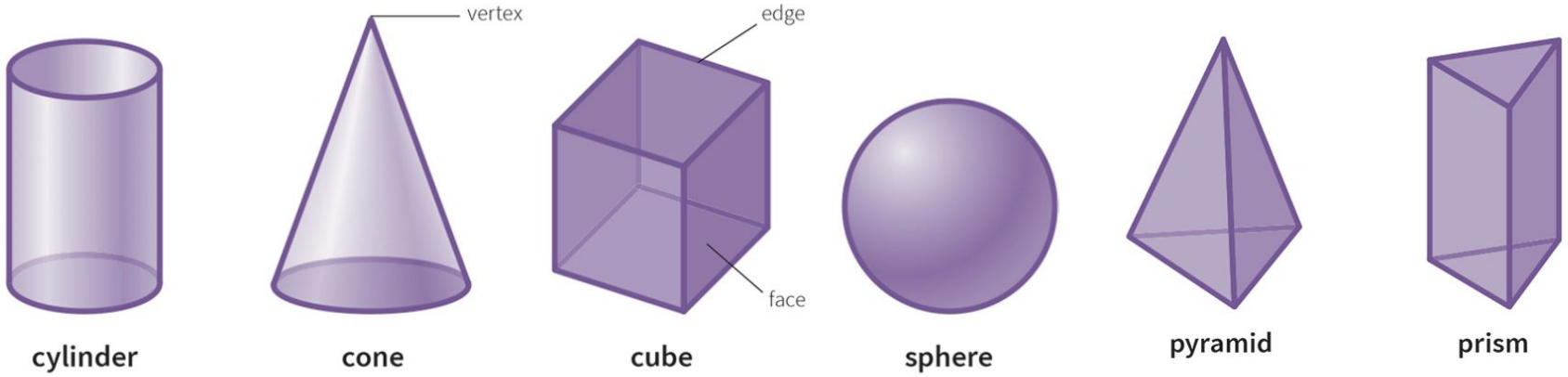
Modelling 3D Objects




```python
# Function to draw a cylinder
def draw_cylinder(radius=1, height=2,
resolution=100):
    z = np.linspace(0, height, resolution)
    theta = np.linspace(0, 2*np.pi, resolution)
    Z, Theta = np.meshgrid(z, theta)
    X = radius * np.cos(Theta)
    Y = radius * np.sin(Theta)
    ax.plot_surface(X, Y, Z, alpha=0.6)
```

```
// Create a cylinder
   private TransformGroup createCylinder() {
        TransformGroup objTransform = new
TransformGroup();
        Cylinder cylinder = new Cylinder(0.5f, 2.0f);
        objTransform.addChild(cylinder);
        return objTransform;
   }
```

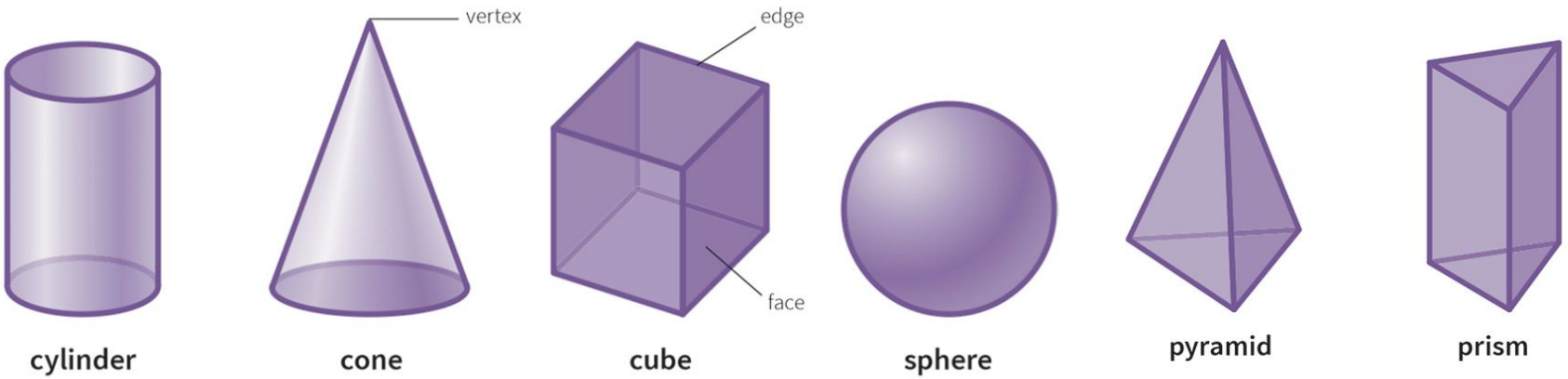
Modelling 3D Objects




```
# Function to draw a cone
def draw_cone(radius=1, height=2, resolution=100):
    z = np.linspace(0, height, resolution)
    theta = np.linspace(0, 2*np.pi, resolution)
    Z, Theta = np.meshgrid(z, theta)
    X = (radius * (height - Z) / height) *
np.cos(Theta)
    Y = (radius * (height - Z) / height) *
np.sin(Theta)
    ax.plot_surface(X, Y, Z, alpha=0.6)
```

```
// Create a cone
    private TransformGroup createCone() {
        TransformGroup objTransform = new
TransformGroup();
        Cone cone = new Cone(0.5f, 2.0f);
        objTransform.addChild(cone);
        return objTransform;
    }
```

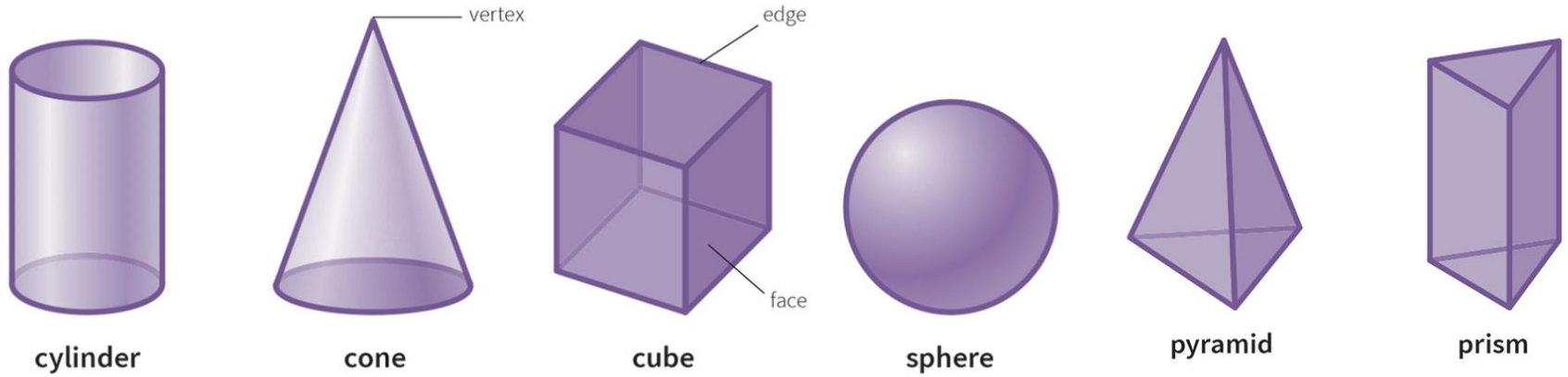
Modelling 3D Objects




```
# Function to draw a sphere
def draw_sphere(radius=1, resolution=100):
    phi = np.linspace(0, np.pi, resolution)
    theta = np.linspace(0, 2 * np.pi,
resolution)
    PHI, THETA = np.meshgrid(phi, theta)
    X = radius * np.sin(PHI) * np.cos(THETA)
    Y = radius * np.sin(PHI) * np.sin(THETA)
    Z = radius * np.cos(PHI)
    ax.plot_surface(X, Y, Z, alpha=0.6)
```

```
// Create a sphere
    private TransformGroup createSphere() {
        TransformGroup objTransform = new
TransformGroup();
        Sphere sphere = new Sphere(1.0f);
        objTransform.addChild(sphere);
        return objTransform;
    }
```

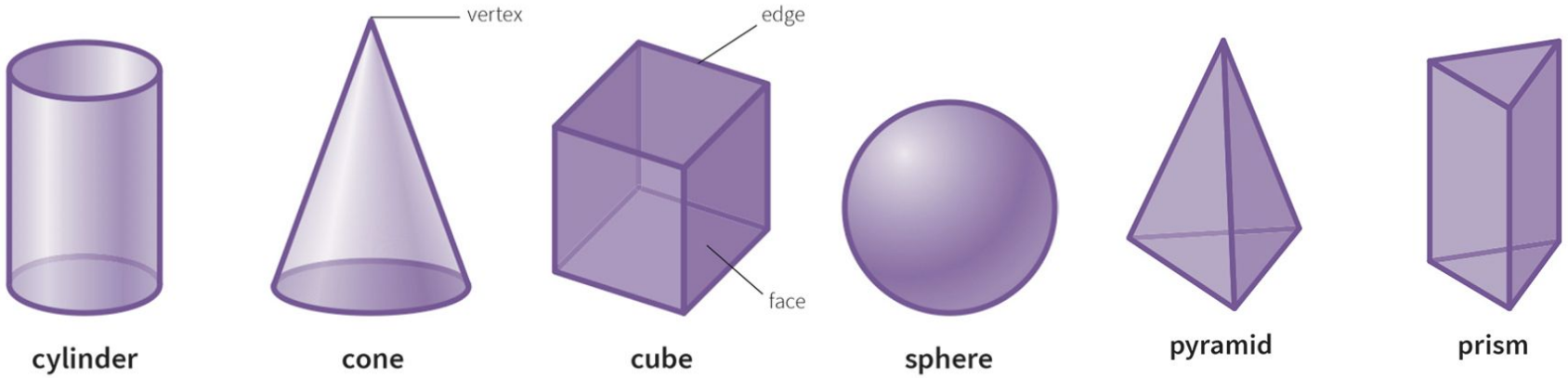
Modelling 3D Objects




```python
# Function to draw a cube
def draw_cube(size=1):
    r = [-size / 2, size / 2]
    X, Y = np.meshgrid(r, r)
    ax.plot_surface(X, Y, r[0] * np.ones_like(X), alpha=0.5)  # Bottom
    ax.plot_surface(X, Y, r[1] * np.ones_like(X), alpha=0.5)  # Top
    ax.plot_surface(X, r[0] * np.ones_like(Y), Y, alpha=0.5)  # Front
    ax.plot_surface(X, r[1] * np.ones_like(Y), Y, alpha=0.5)  # Back
    ax.plot_surface(r[0] * np.ones_like(X), X, Y, alpha=0.5)  # Left
    ax.plot_surface(r[1] * np.ones_like(X), X, Y, alpha=0.5)  # Right
```

```java
// Create a cube
    private TransformGroup
createCube() {
        TransformGroup objTransform =
new TransformGroup();
        Box box = new Box(1.0f, 1.0f,
1.0f, null);
        objTransform.addChild(box);
        return objTransform;
    }
```

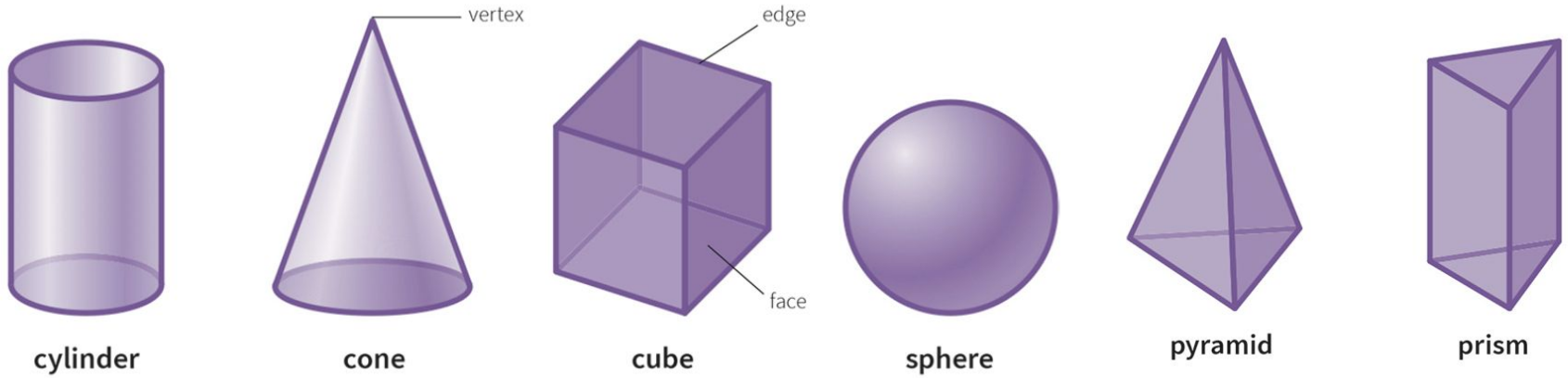
Modelling 3D Objects




```
# Function to draw a pyramid
def draw_pyramid(base_size=1, height=2,
resolution=4):
    half = base_size / 2
    X = np.array([[-half, half, half, -half, 0]])
    Y = np.array([[-half, -half, half, half, 0]])
    Z = np.array([[0, 0, 0, 0, height]])
    ax.plot_trisurf(X[0], Y[0], Z[0], alpha=0.7)
```

```
// Create a pyramid (simple cone with square base)
    private TransformGroup createPyramid() {
        TransformGroup objTransform = new
TransformGroup();
        Cone pyramid = new Cone(0.5f, 2.0f); //
Simplified pyramid
        objTransform.addChild(pyramid);
        return objTransform;
    }
```

Modelling 3D Objects




```
# Function to draw a prism
def draw_prism(base_size=1,
height=2, resolution=4):
    draw_cube(base_size)
    ax.set_zlim(0, height)
```

```
// Create a prism (simplified box for demonstration)
    private TransformGroup createPrism() {
        TransformGroup objTransform = new TransformGroup();
        Box prism = new Box(1.0f, 1.0f, 2.0f, null);
        objTransform.addChild(prism);
        return objTransform;
    }
```

Modelling 3D Objects

What Kind of Objects are Considered in Graphics?

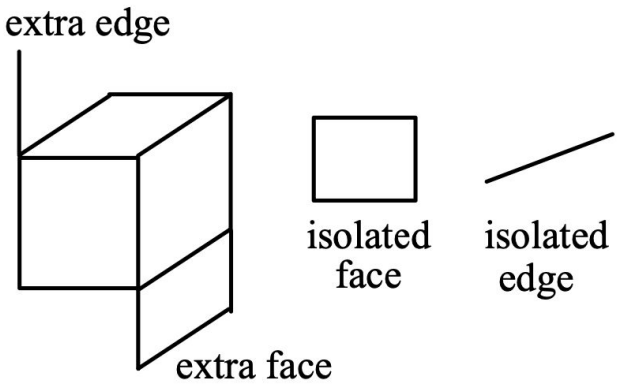
In principle, any subset of space $\mathbb{R}^3$ could be seen as a three-dimensional object. However, this means that even points, edges, or planes are considered as 3D

Modelling 3D Objects

What Kind of Objects are Considered in Graphics?

In principle, any subset of space $\mathbb{R}^3$ could be seen as a three-dimensional object. However, this means that even points, edges, or planes are considered as 3D

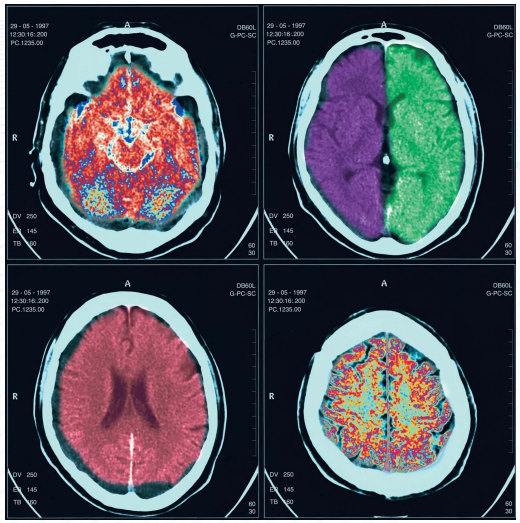
The figure shows some examples of **how 3D objects should not look like**. Isolated or dangling edges and faces should be avoided



Modelling 3D Objects

What Kind of Objects are Considered in Graphics?

For the purpose of showing a 3D object, its surface is of importance and **not the set of points that are occupied** by the object



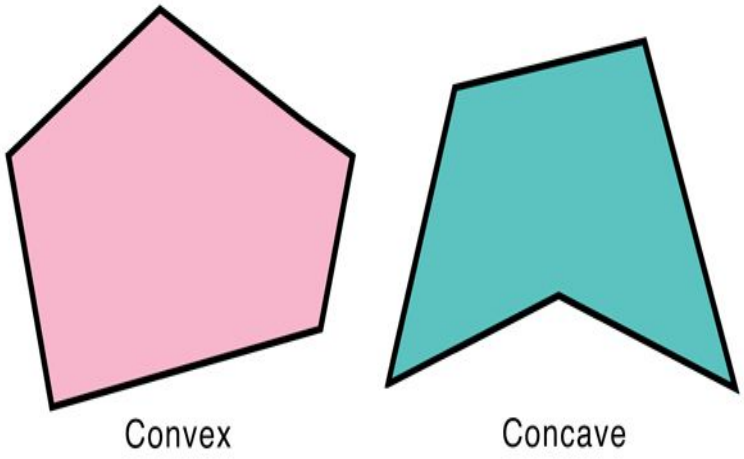
In certain applications, no explicit definition of the surface of the object exists. In such cases, it is very common that the object is **first described as a set of points and then the surface is derived from this set**

Modelling 3D Objects

Dealing with Complex Shapes

For complex shapes like round and bent shapes, there are various sophisticated techniques. However, **these models are usually not taken directly** for the generation of an image in a scene.

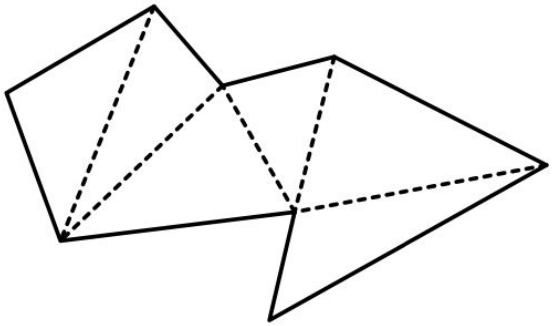
Instead, **surfaces are usually approximated by a larger number of polygons**, triangles in most cases, in order to simplify computations for illumination and projection



Modelling 3D Objects

Dealing with Complex Shapes

For arbitrary surfaces, it **might be even impossible to find an analytical expression** for the representation of the projection. Efficient and fast computation would become impossible as opposed to using polygons.



The **approximation of a curved surface by polygons is called tesselation**. Using only triangles is no real restriction since any polygon can be partitioned into triangles

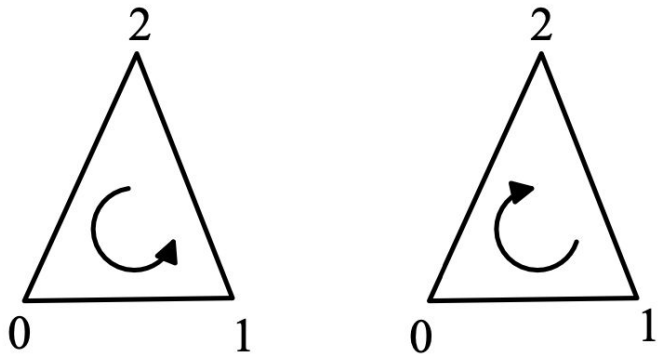
Modelling 3D Objects

Orientation of Triangles and Polygons

Single triangles or polygons for modelling a surface are usually oriented in order to **determine which side of the polygon is outside of the surface**.

The orientation is given by the order of the polygon's vertices. The vertices are listed in **counterclockwise order when looking onto the surface from the outside of an object**

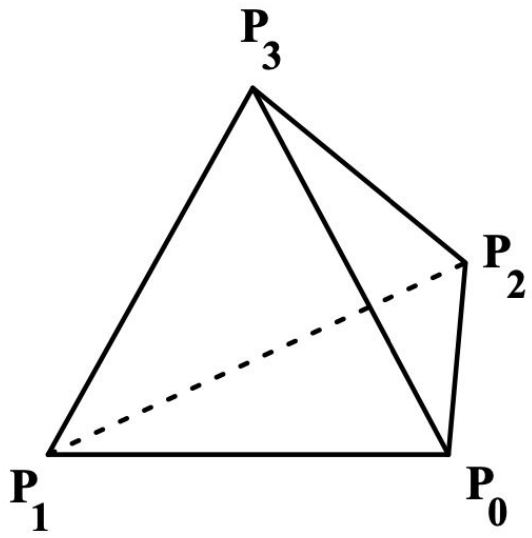
Orientation speeds up rendering since it **can ignore surfaces** the do not point to the viewer



Viewable Hidden

Modelling 3D Objects

Orientation of Triangles and Polygons



How many faces of the tetrahedron are there?

Describe these faces using the vertices with orientation

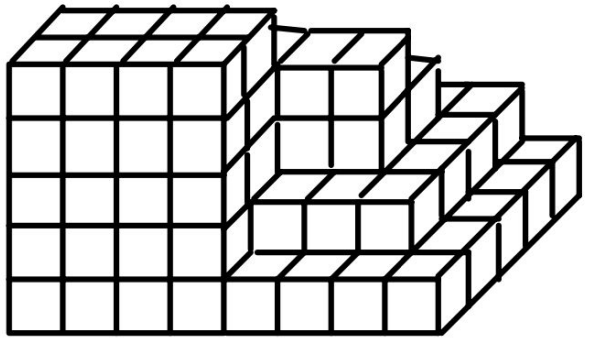
Modelling Techniques

Voxels: 3D Building Blocks

It is a very simple technique for modelling 3D objects. **The 3D space is partitioned into a grid of small, equisized cubes**. It is a counterpart of a 2D grid.

A three-dimensional object is defined by those **voxels that lie within the interior** of the object.

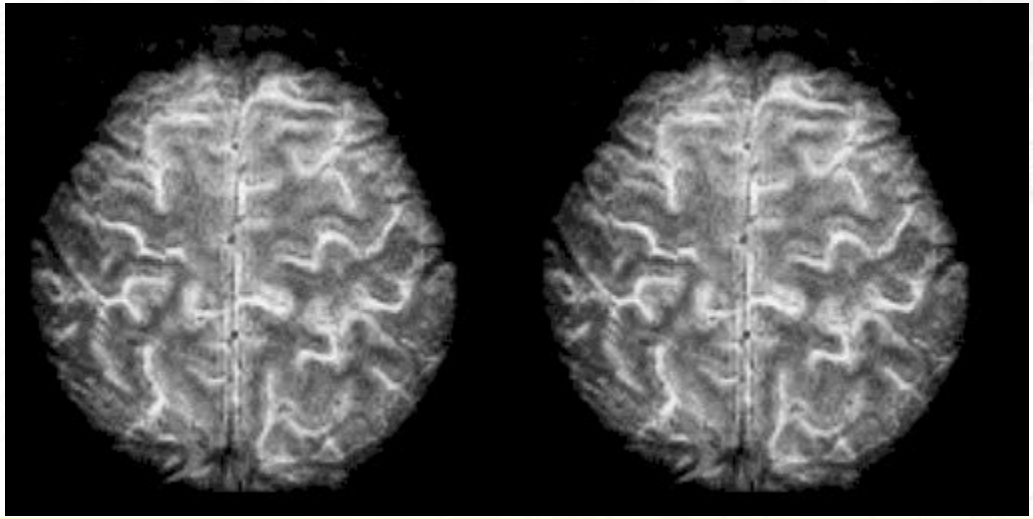
The **computational costs in terms of memory and time for handling voxel models can be enormous**. If 2D objects can be as big as 100 x 100, 10,000 pixels are needed. Then 3D objects can be as huge as 100 x 100 x 100 = 10^6 voxels are needed



Modelling Techniques

Voxels: 3D Building Blocks

They are suitable for **modelling objects based on tomography data**, which provide information about the tissue density inside the measured body or object.



Modelling Techniques

Voxels: 3D Building Blocks

Advantages

Voxels are a more “accurate” 3D building block than any other modeling type, as they mimic particles

Voxels are the quickest way to quickly model and visualize volumetric data (especially natural/organic formations)

Disadvantages

Without using prohibitively expensive techniques, it is much harder to build complex objects using voxels

Current computer hardware is optimized for rendering polygons

Lecture References:

CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu
https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/
https://www.csd.uwo.ca/~oveksler/Courses/CS434a_541a/Lecture16.pdf