3D Computer Graphics (II)

Multimedia Techniques & Applications

Yu-Ting Wu

(with slides borrowed from Prof. Yung-Yu Chuang)

1

What is Computer Graphics

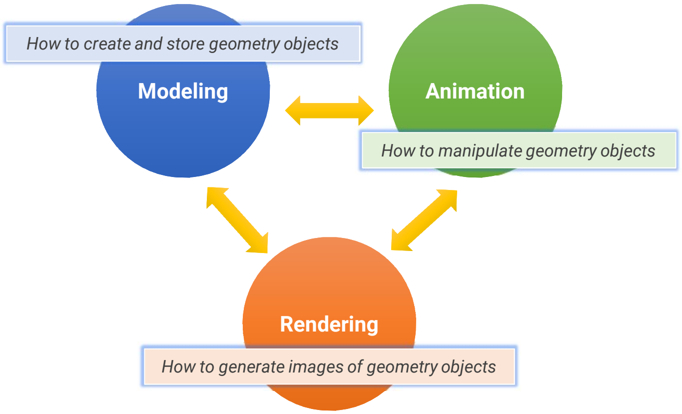
- Computer graphics are pictures and films created using computers
- Computer graphics is the process of creation, storage and manipulation of models and images using data structure and algorithms




we will focus on this


2

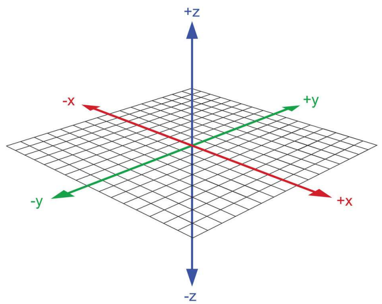
Major Subfields of Computer Graphics



3

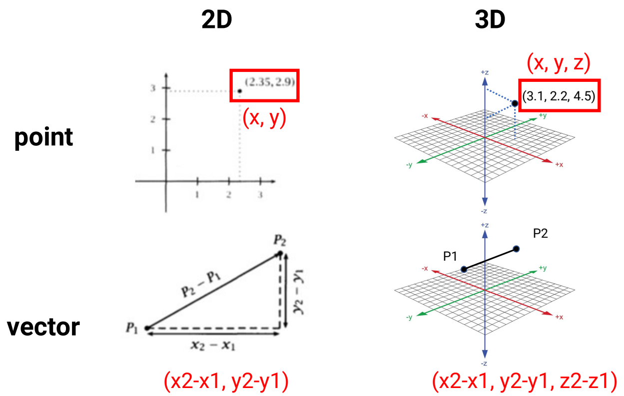
Description of a 3D World

- **3D coordinate system**
  - The formation of x, y, and z axis can be different



4

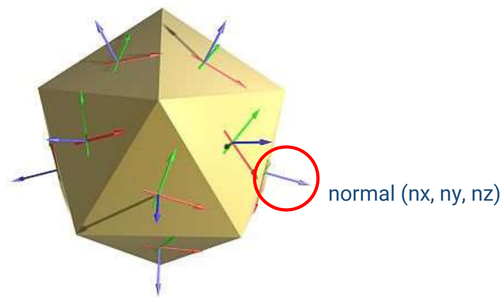
From 2D to 3D



5

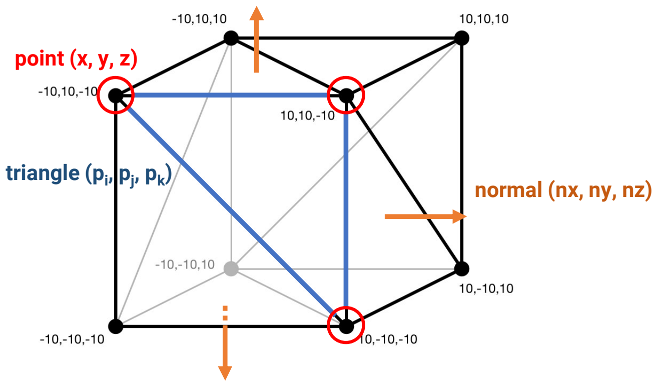
Surface Normal

- A **surface normal** is a vector that is **perpendicular** to a surface at a particular position



6

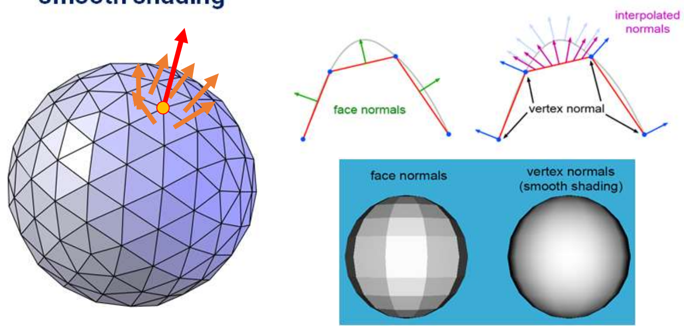
Points, Normals, and Triangles



7

Vertex Normal

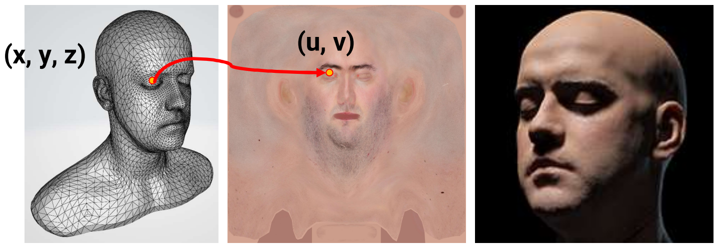
- Sometimes we also define the normal of a vertex by averaging the surface normals of its adjacent faces for ***smooth shading***



8

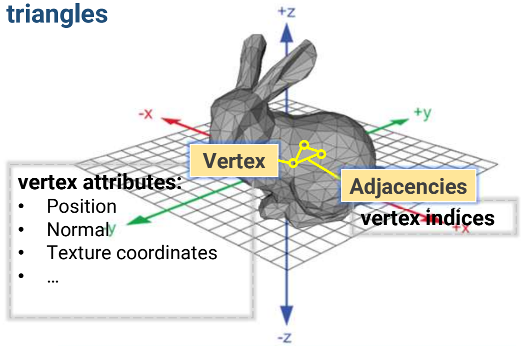
Texture Coordinate

- **Texture**
  - Used to represent spatially-varying data
  - Decouple materials from geometry
  - Need a mapping from 3D (xyz, object space) to 2D (uv, image space), called **texture coordinate**



Description of an 3D Object

- We will focus on triangle mesh today
- Define the **position**, **normal**, and **texture coordinate** on the **vertices of triangles**



Common 3D Model Format

- **Wavefront (*.obj)**
- Polygon file format (*.ply)
- **Filmbox (*.fbx)**
- MAX (*.max)
- Digital Asset Exchange File (*.dae)
- STereoLithography (*.stl)

Wavefront (OBJ)

```
mtllib default.mtl
```
material file

```
v 0.5 0.5 -0.5
v 0.5 0.5 0.5
v -0.5 0.5 0.5
v -0.5 0.5 -0.5
v 0.5 -0.5 -0.5
v 0.5 -0.5 0.5
v -0.5 -0.5 0.5
v -0.5 -0.5 -0.5
```
vertex position

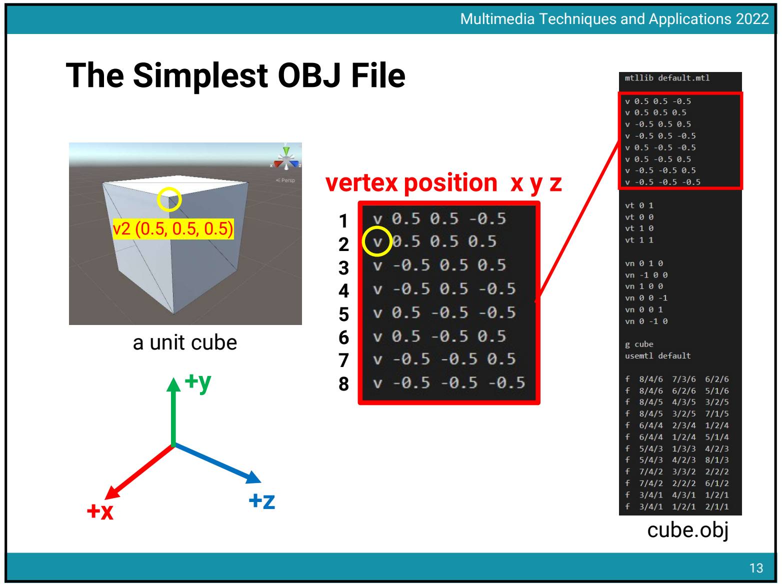
```
vt 0 1
vt 0 0
vt 1 0
vt 1 1
```
vertex texture coordinate

```
vn 0 1 0
vn -1 0 0
vn 1 0 0
vn 0 0 -1
vn 0 0 1
vn 0 -1 0
```
vertex normal

```
g cube
usemtl default
```
group name
group material

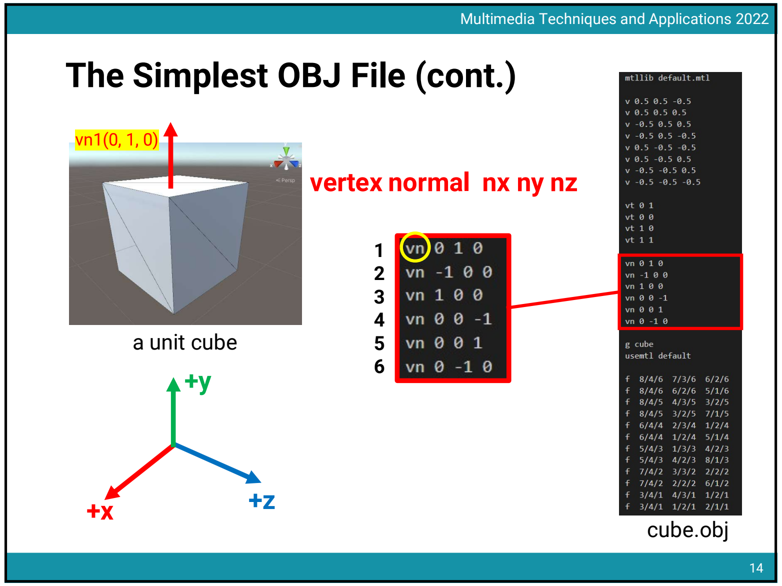
```
f  8/4/6  7/3/6  6/2/6
f  8/4/6  6/2/6  5/1/6
f  8/4/5  4/3/5  3/2/5
f  8/4/5  3/2/5  7/1/5
f  6/4/4  2/3/4  1/2/4
f  6/4/4  1/2/4  5/1/4
f  5/4/3  1/3/3  4/2/3
f  5/4/3  4/2/3  8/1/3
f  7/4/2  3/3/2  2/2/2
f  7/4/2  2/2/2  6/1/2
f  3/4/1  4/3/1  1/2/1
f  3/4/1  1/2/1  2/1/1
```
face adjacencies

cube.obj



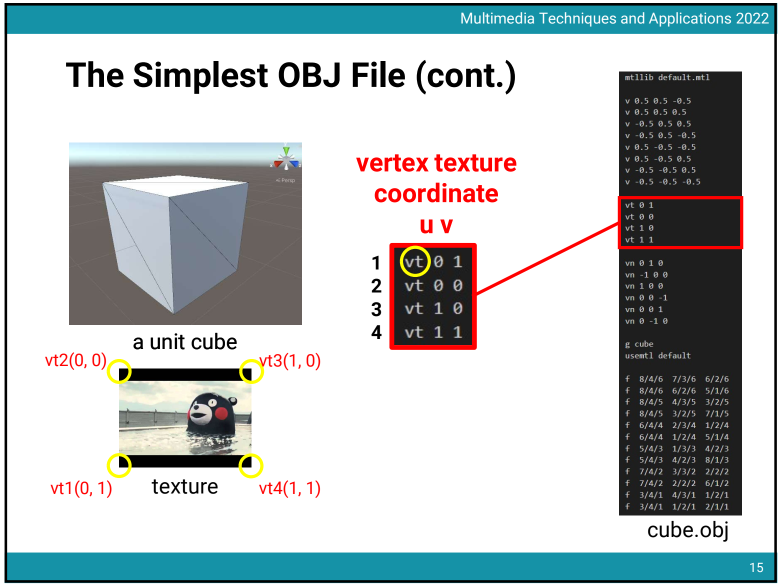
Multimedia Techniques and Applications 2022
The Simplest OBJ File
v2 (0.5, 0.5, 0.5)
a unit cube
+y
+x
+z
vertex position x y z
1 v 0.5 0.5 -0.5
2 v 0.5 0.5 0.5
3 v -0.5 0.5 0.5
4 v -0.5 0.5 -0.5
5 v 0.5 -0.5 -0.5
6 v 0.5 -0.5 0.5
7 v -0.5 -0.5 0.5
8 v -0.5 -0.5 -0.5
cube.obj
13

13



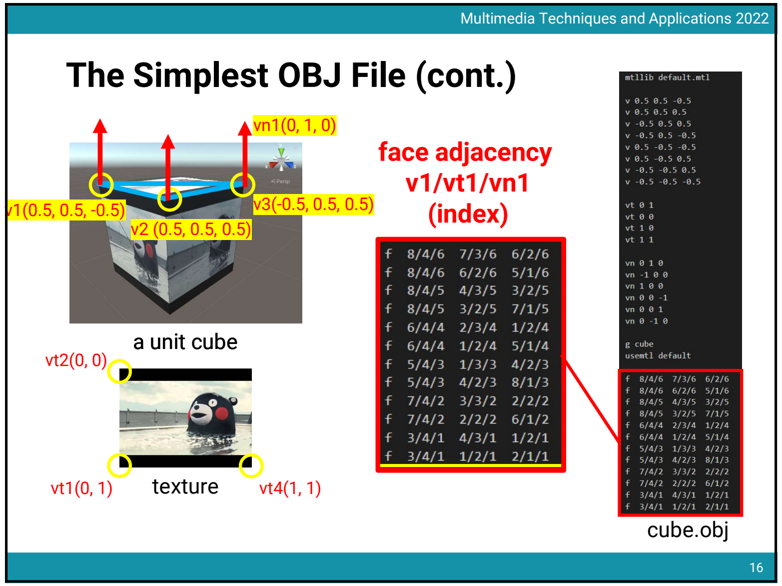
Multimedia Techniques and Applications 2022
The Simplest OBJ File (cont.)
vn1(0, 1, 0)
a unit cube
+y
+x
+z
vertex normal nx ny nz
1 vn 0 1 0
2 vn -1 0 0
3 vn 1 0 0
4 vn 0 0 -1
5 vn 0 0 1
6 vn 0 -1 0
cube.obj
14

14



Multimedia Techniques and Applications 2022
The Simplest OBJ File (cont.)
a unit cube
vertex texture coordinate u v
1 vt 0 1
2 vt 0 0
3 vt 1 0
4 vt 1 1
vt2(0, 0)
vt3(1, 0)
vt1(0, 1)
texture
vt4(1, 1)
cube.obj
15

15



Multimedia Techniques and Applications 2022
The Simplest OBJ File (cont.)
vn1(0, 1, 0)
v1(0.5, 0.5, -0.5)
v2 (0.5, 0.5, 0.5)
v3(-0.5, 0.5, 0.5)
a unit cube
face adjacency v1/vt1/vn1 (index)
f 8/4/6 7/3/6 6/2/6
f 8/4/6 6/2/6 5/1/6
f 8/4/5 4/3/5 3/2/5
f 8/4/5 3/2/5 7/1/5
f 6/4/4 2/3/4 1/2/4
f 6/4/4 1/2/4 5/1/4
f 5/4/3 1/3/3 4/2/3
f 5/4/3 4/2/3 8/1/3
f 7/4/2 3/3/2 2/2/2
f 7/4/2 2/2/2 6/1/2
f 3/4/1 4/3/1 1/2/1
f 3/4/1 1/2/1 2/1/1
vt2(0, 0)
vt1(0, 1)
texture
vt4(1, 1)
cube.obj
16

16

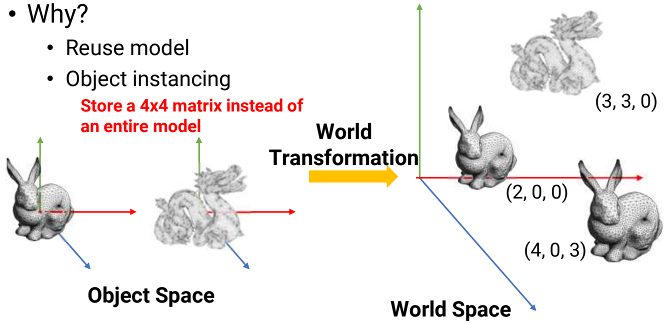
Play with Mesh Viewer

- There are lots of free models on the internet

- Try to download one and play with it in the viewer
- Free mesh viewer
  - Windows 3D Viewer
  - Meshlab https://www.meshlab.net/
  - Blender (modeling tool) https://www.blender.org/
  - Unity (game engine)
  - Unreal engine (game engine)

Object Space and World Space

- Shapes (or objects) are defined in **object space** and transformed to **world space**
- Why?
  - Reuse model
  - Object instancing



Object Space and World Space (cont.)

- Demo with Unity

Recap: 2D Transformation

- Using **3x3 matrix** allows us to perform all transformations using matrix/vector multiplications
  - We can also **pre-multiply** all the matrices together
- We call the (x, y, 1) representation for (x, y) **homogeneous coordinate**

Translation
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scaling
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

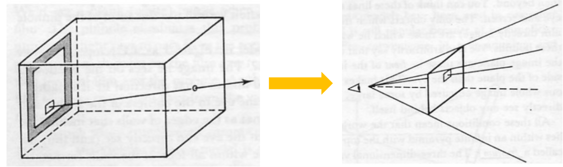
3D Transformation

- A 3D transformation ***T*** is represented as a **4x4 matrix**, with **homogeneous coordinate**

	2D	3D
Translation	$\begin{bmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 0 & t_x\\ 0 & 1 & 0 & t_y\\ 0 & 0 & 1 & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}$
Scaling	$\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}s_x & 0 & 0 & 0\\ 0 & s_y & 0 & 0\\ 0 & 0 & t_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$
Rotation	$\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}cos(v) & -sin(v) & 0 & 0\\ sin(v) & cos(v) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$

(Virtual) Camera

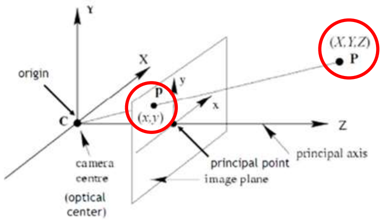
- For most cases in computer graphics, we use a **perspective pinhole camera model** for its simplicity

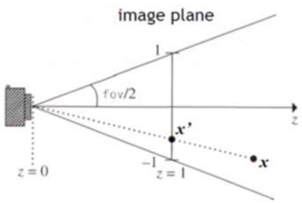


The virtual film is placed **in front of** the camera for avoiding **up-side-down** image

Perspective Pinhole Camera Model

- Assume the camera is located at **origin** and look to **+Z** (or another axis depends on the graphics system)

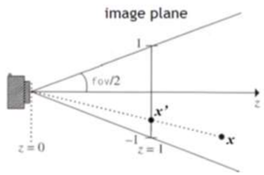


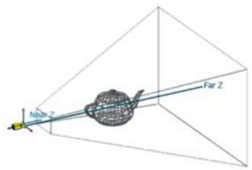


Perspective Pinhole Camera Model (cont.)

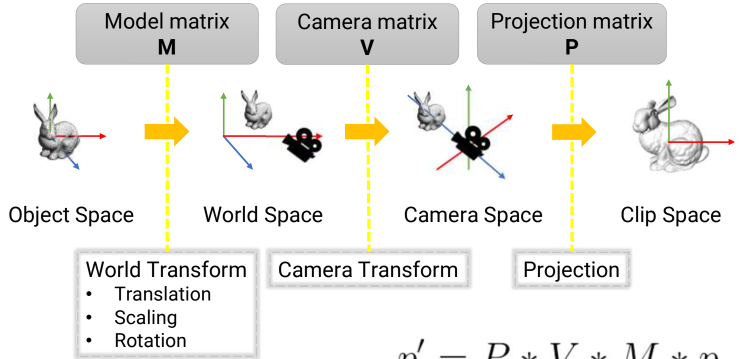
- 3D points can be projected on the virtual film by using a projection matrix

$$\begin{bmatrix}\tan^{-1}(\frac{\mathrm{FOV}_x}{2}) & 0 & 0 & 0\\ 0 & \tan^{-1}(\frac{\mathrm{FOV}_y}{2}) & 0 & 0\\ 0 & 0 & \frac{Z_{far}}{Z_{far}-Z_{near}} & -\frac{Z_{far}Z_{near}}{Z_{far}-Z_{near}}\\ 0 & 0 & 1 & 0\end{bmatrix}$$





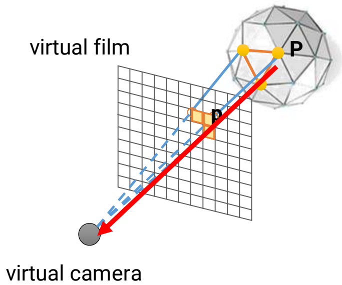
Transform between Spaces

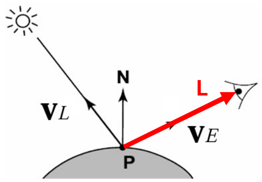


$$p' = P * V * M * p$$

Shading

- Simulate the **interaction** between a **light** and a **surface point**

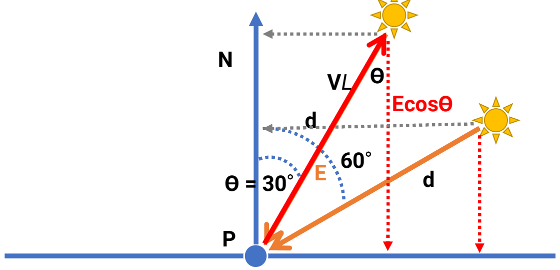




Point **P** on a surface through a pixel **p**
Normal **N** at **P**
Lighting direction **V***L*
Viewing direction **V***E*
Goal: compute color **L** for pixel **p**

Lambertian Term

- Assume the two lights have equal intensity and equal distance to **P**, which light can contribute more to the point **P**?
  - The contribution is proportional to **cosθ**, where **θ** is the angle between surface normal **N** and lighting direction **V***L*



Virtual Lights in Computer Graphics

- Point light
- Spot light
- Area light

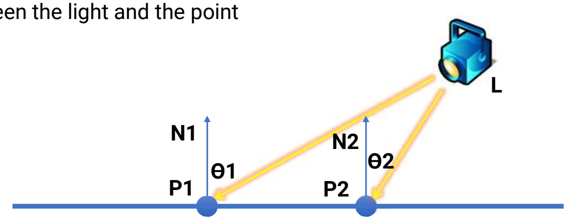
} local lights

- Directional light
- Environment light

} distant lights

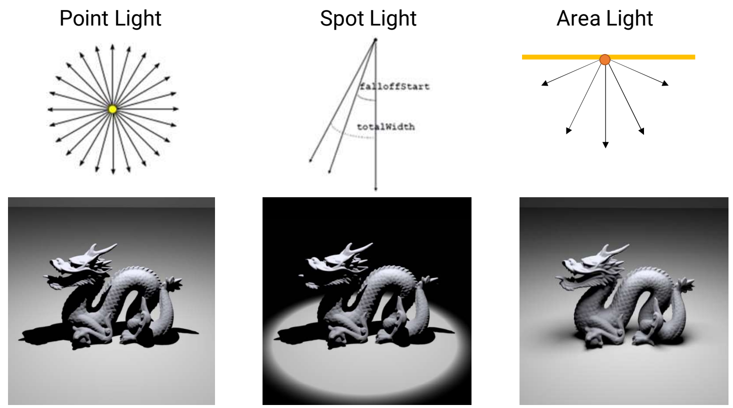
Local Lights

- The distance between a light and a surface is **not** long enough compared to the scene scale
- The position of a light need to be taken into account during shading
  - **Lighting direction = |L – P|**
  - **Lighting attenuation** is proportional to the square of distance between the light and the point



29

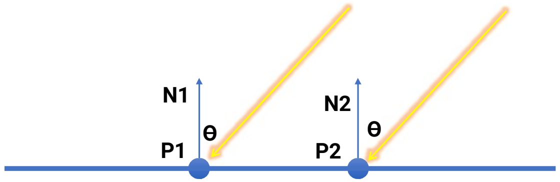
Local Lights (cont.)



30

Distant Lights

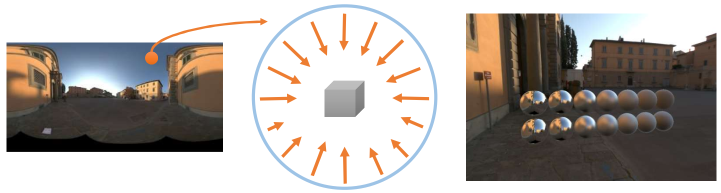
- The distance between a light and a surface is long enough compared to the scene scale and **can be ignored**
  - **Lighting direction** is **fixed**
  - **No lighting attenuation**
- **Directional light (sun)** is the most common distant light



31

Environment Light

- Environment light illuminates the scene from a **virtual sphere at infinite distance**
- The spherical energy distribution is usually represented with longitude-latitude images
- Also called **image-based lighting (IBL)**



32

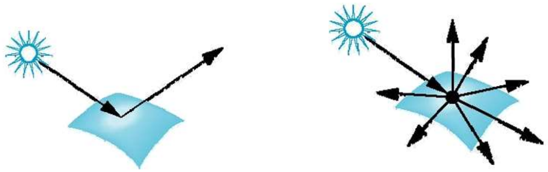
Environment Light (cont.)

- Widely used in digital visual effects and film production

33

Materials

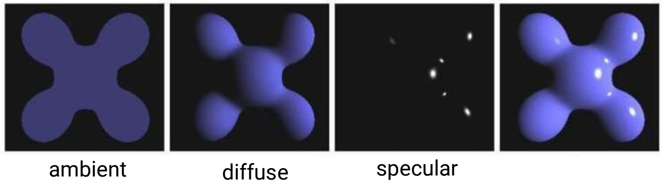
- **Surface types**
  - The **smoother** a surface, the more reflected light is concentrated in the direction a **perfect mirror** would reflect the light



34

Basics of Local Shading

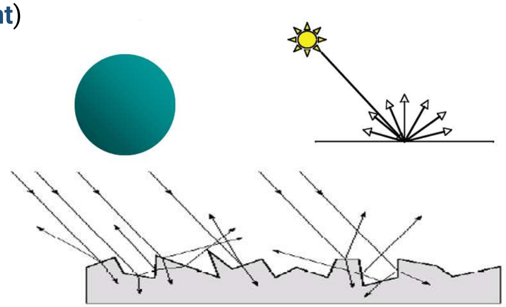
- **Diffuse reflection**
  - Light goes everywhere; colored by object color
- **Specular reflection**
  - Happens only near mirror configuration; usually white
- **Ambient reflection**
  - Constant accounted for other source of illumination



35

Diffuse Shading

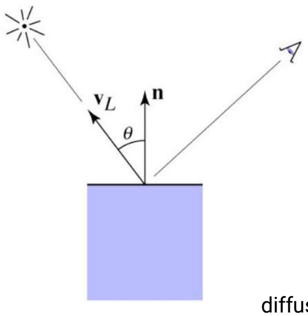
- Assume light reflects **equally in all directions**
  - The surface is rough with lots of tiny microfacets
- Therefore, surface looks same color from all views (**view independent**)



36

Diffuse Shading (cont.)

- Applies to diffuse, Lambertian or matte surface



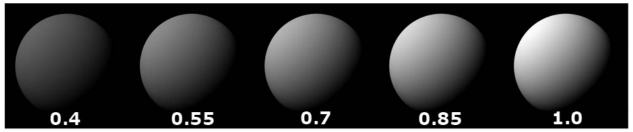
illumination from source

$$L_d = k_d \; I \; \max(0, n \cdot V_L)$$

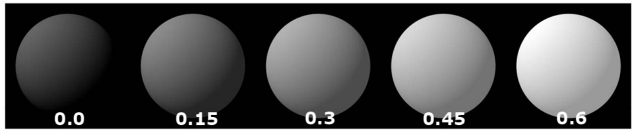
diffuse coefficient

diffusely reflected light

Diffuse Shading (cont.)



diffuse-reflection model with different k_d



ambient and diffuse-reflection model with different k_a

$I_a = 1.0 \quad k_d = 0.4$

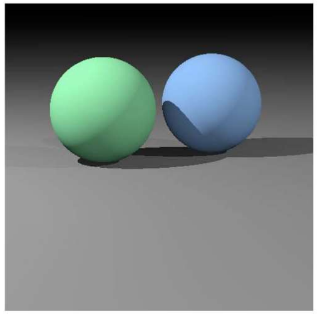
Diffuse Shading (cont.)

- For color objects, apply the formula for each color channel separately
- Light can also be non-white

Example:
white light: (0.9, 0.9, 0.9)
yellow light: (0.8, 0.8, 0.2)

$$L_d = \boxed{k_d}\,\boxed{I}\,\max(0, n \cdot V_L)$$

Example:
green ball: (0.2, 0.7, 0.2)
blue ball: (0.2, 0.2, 0.7)

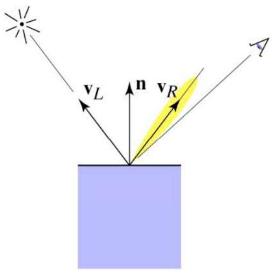


Specular Shading

- Some surfaces have highlights, mirror-like reflection
- **View direction dependent**
- Especially obvious for smooth shinny surfaces

Specular Shading (cont.)

- Also known as glossy
- Phong specular model [1975]

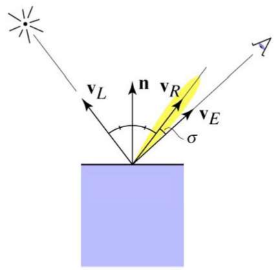


$$V_R = V_L + 2((\mathbf{n} \cdot V_L)\,\mathbf{n} - V_L)$$
$$= 2(\mathbf{n} \cdot V_L)\,\mathbf{n} - V_L$$

41

Specular Shading (cont.)

- Also known as glossy
- Phong specular model [1975]
  - Fall off gradually from the perfect reflection direction



$$V_R = V_L + 2((\mathbf{n} \cdot V_L)\,\mathbf{n} - V_L)$$
$$= 2(\mathbf{n} \cdot V_L)\,\mathbf{n} - V_L$$

$$L_s = k_s \; I \; \max(0, \cos\sigma)^n$$
$$= k_s \; I \; \max(0, V_E \cdot V_R)^n$$

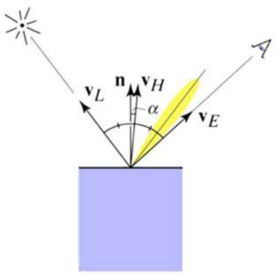
specular coefficient

specularly reflected light

42

Specular Shading (cont.)

- Phong variant: **Blinn-Phong**
  - Rather than computing reflection directly; just compare to normal bisection property
  - One can prove $\cos^n(\sigma) = \cos^{4n}(\alpha)$



$$V_H = \text{bisector}(V_L, V_E)$$
$$= \frac{(V_L + V_E)}{\|V_L + V_E\|}$$

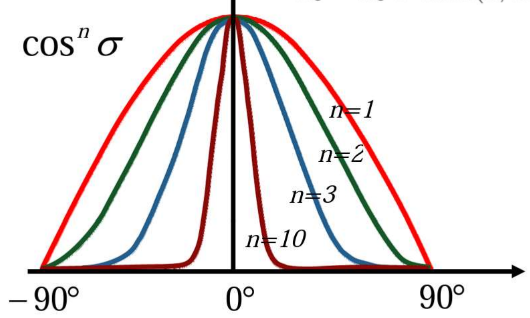
$$L_s = k_s \; I \; \max(0, \cos\alpha)^n$$
$$= k_s \; I \; \max(0, \mathbf{n} \cdot V_H)^n$$

43

Specular Shading (cont.)

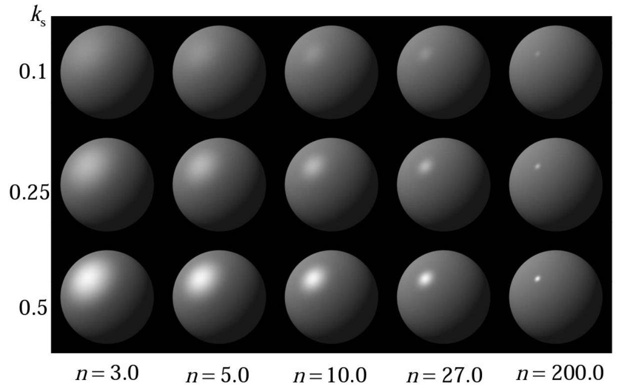
- Increase n narrows the lobe

$$L_s = k_s \; I \; \max(0, \cos\alpha)^n$$

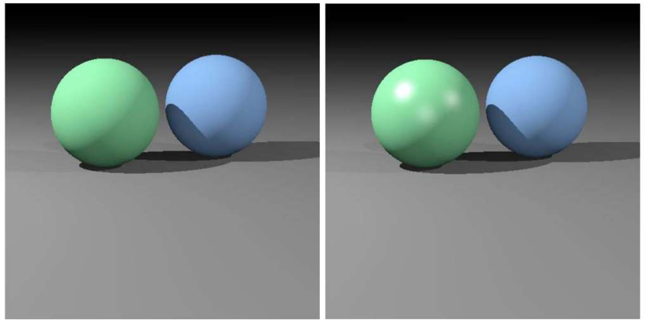


44

Specular Shading (cont.)



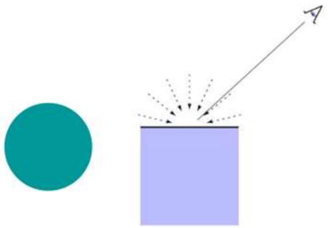
Specular Shading (cont.)



diffuse | diffuse + specular

Ambient Shading

- Add constant color to account for disregarded illumination and fill in black shadows
- A cheap hack



ambient light

$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

Put it All Together

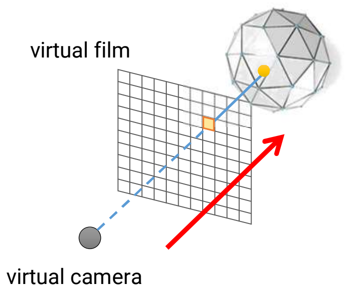
- Compute the contribution of a light to a point by including **ambient**, **diffuse**, and **specular** components

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + I(k_d \max(0, \mathbf{n} \cdot V_L) + k_s \max(0, \mathbf{n} \cdot V_H)^n) \end{aligned}$$

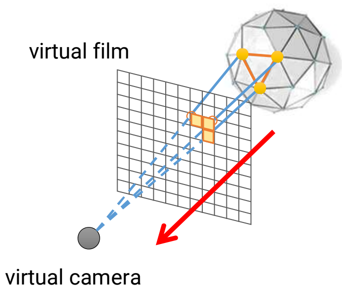
- If there are many lights, just sum over all the lights because lighting is **linear**

Bring Triangles into Pixels

Ray Tracing



Rasterization

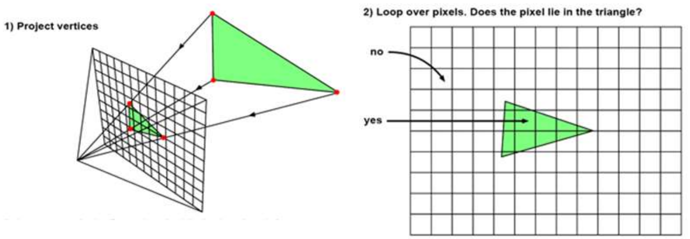


Graphics API (or Library)

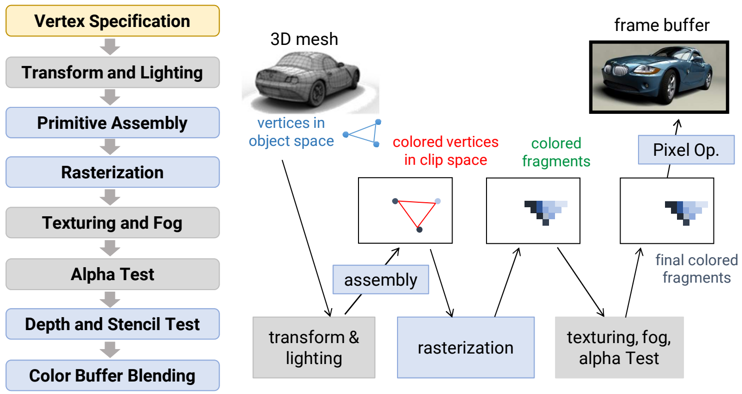
- A program library designed to aid in rendering computer graphics to a monitor
- Typically involves providing optimized versions of functions that handle common **rendering tasks**
- **Rasterization-based**
- Common graphics APIs are
  - OpenGL
  - OpenGL ES
  - WebGL
  - DirectX
  - Metal
  - Vulkan

Rasterization

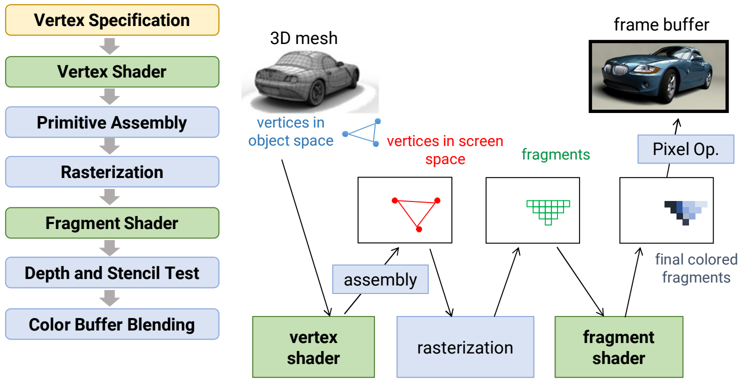
- Rasterization
  - Bring the **triangles** to **pixels**
  - Determine which pixels are covered by a projected 3D triangle



OpenGL 1.X (Fixed Function) Graphics Pipeline

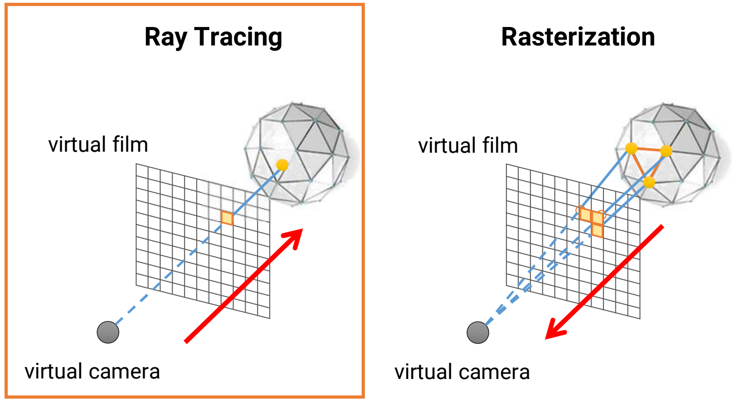


OpenGL 2.0 Graphics Pipeline



53

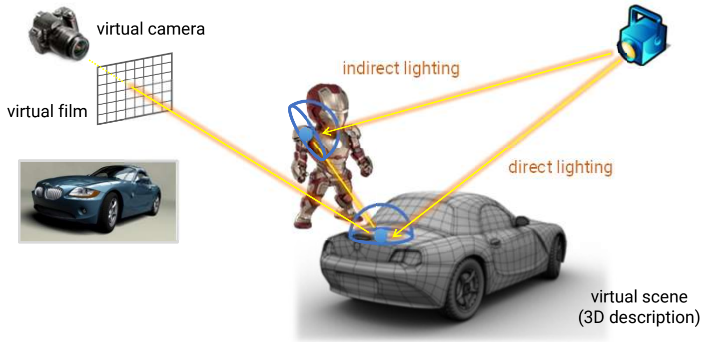
Bring Triangles into Pixels



54

Ray Tracing

- Simulate a wide variety of light transport paths by tracing rays and calculate their carried energy

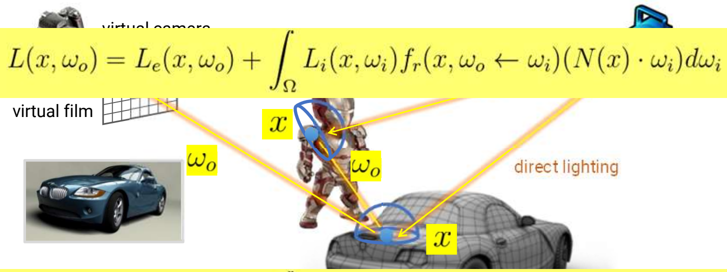


55

Ray Tracing

- Simulate a wide variety of light transport paths by tracing rays and calculate their carried energy

$$L(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega L_i(x,\omega_i) f_r(x,\omega_o \leftarrow \omega_i)(N(x)\cdot\omega_i)d\omega_i$$

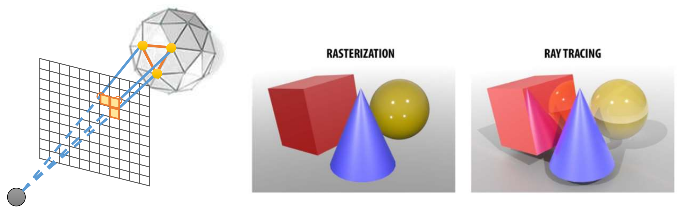


$$L(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega L_i(x,\omega_i) f_r(x,\omega_o \leftarrow \omega_i)(N(x)\cdot\omega_i)d\omega_i$$

56

Rasterization v.s. Ray Tracing

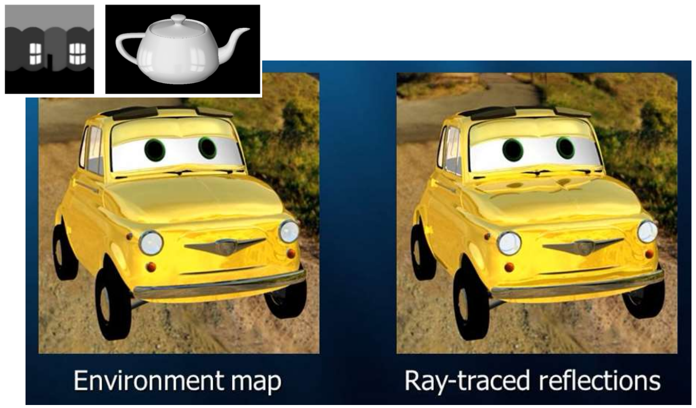
- Rasterization is more friendly to hardware and usually has higher parallelism
- But it is more difficult to simulate effects such as reflection, refraction, shadows, and global illumination
  - Need specialized algorithms



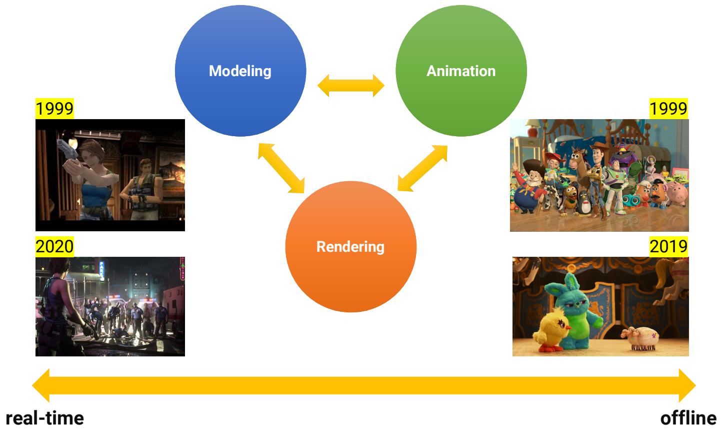
Rasterization v.s. Ray Tracing

- Ray tracing is more general
- However, its simulator usually has a slow convergence rate and produces lots of noises when samples are not enough

Rasterization v.s. Ray Tracing



Real-time v.s. Offline Graphics



Multimedia Techniques and Applications 2022

Real-time Ray Tracing

- FIRST DAY: A Star Wars short film made with UE5

61

61

Multimedia Techniques and Applications 2022

How to Learn 3D Computer Graphics?

- Online materials
  - https://ogldev.org/
  - http://www.opengl-tutorial.org/
  - https://learnopengl.com/
  - https://antongerdelan.net/opengl/
- Or …
- Come to my class ***"Introduction to Computer Graphics"*** **next semester**!

62

62