Matrices

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

- A matrix is a rectangular array of numbers.
- A general matrix will be represented by an upper-case italicised letter.
- The element on the *i*th row and *j*th column is denoted by $a_{i,j}$. Note that we start indexing at 1, whereas *C* indexes arrays from 0.

Matrices - addition

- Given two matrices *A* and *B* if we want to add *B* to *A* (that is form *A+B*) then if *A* is (*n*×*m*), *B* must be (*n*×*m*), Otherwise, *A+B* is not defined.
- The addition produces a result, *C* = *A+B*, with elements:

$$C_{i,j} = A_{i,j} + B_{i,j}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

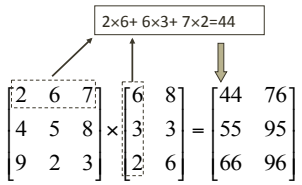
Matrices - Multiplication

- Given two matrices *A* and *B* if we want to multiply *B* by *A* (that is form *AB*) then if *A* is (*n*×*m*), *B* must be (*m*×*p*), i.e., the number of columns in *A* must be equal to the number of rows in *B*. Otherwise, *AB* is not defined.
- The multiplication produces a result, *C* = *AB*, with elements:

$$C_{i,j} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

- (Basically we multiply the first row of *A* with the first column of *B* and put this in the $c_{1,1}$ element of *C*. And so on…).

Matrices - Multiplication



$$\begin{bmatrix} 2 & 6 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 6 & 8 \\ 3 & 3 \\ 2 & 6 \end{bmatrix}$$

Undefined!
2x2 x 3x2 2!=3

2x2 x 2x4 x 4x4 is allowed. Result is 2x4 matrix

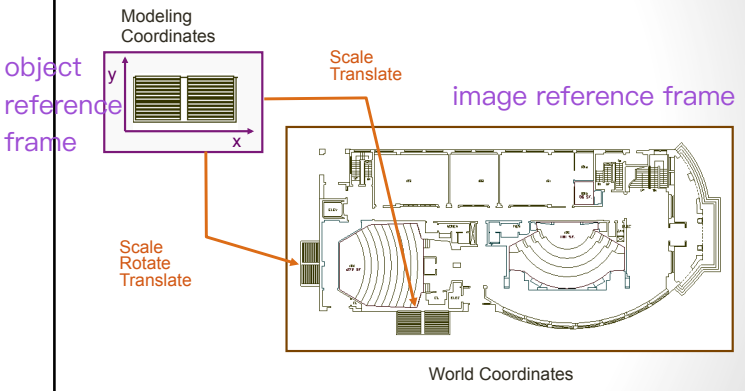
Matrices - basics

- Unlike scalar multiplication, $AB \neq BA$
- Matrix multiplication distributes over addition:

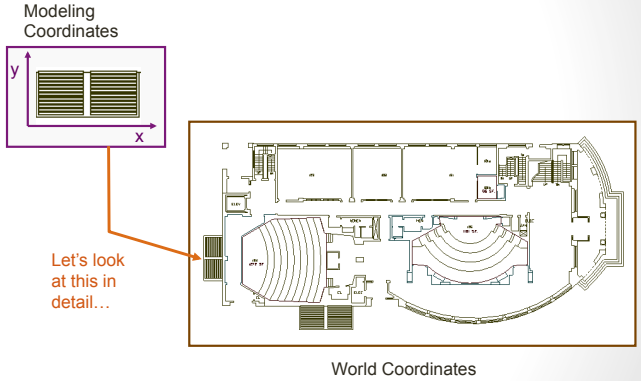
  $A(B+C) = AB + AC$
- Identity matrix for multiplication is defined as I.
- The transpose of a matrix, A, is either denoted A^T or A' is obtained by swapping the rows and columns of A:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \Rightarrow A' = \begin{bmatrix} a_{1,1} & a_{2,1} \\ a_{1,2} & a_{2,2} \\ a_{1,3} & a_{2,3} \end{bmatrix}$$

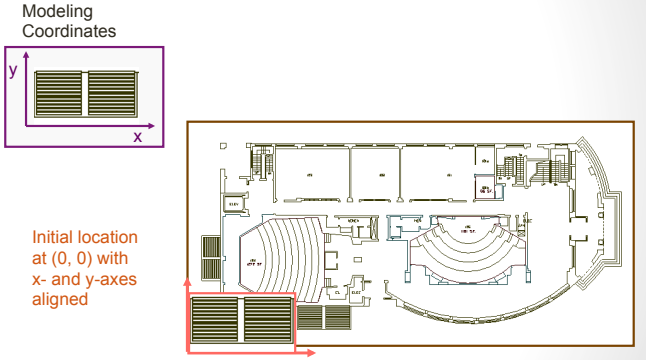
2D Modeling Transformations



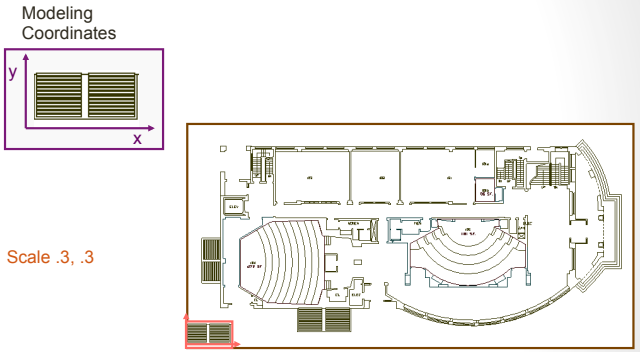
2D Modeling Transformations



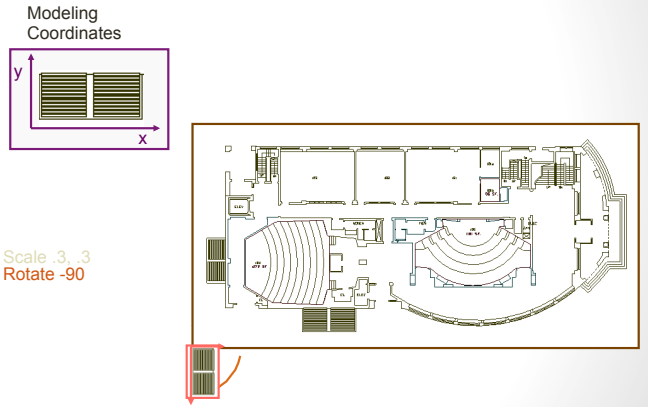
2D Modeling Transformations



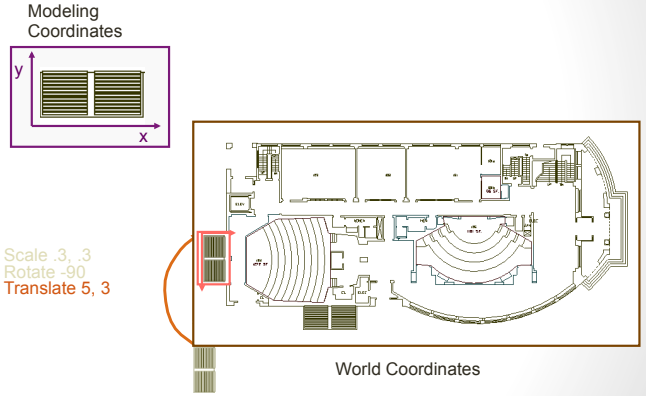
2D Modeling Transformations



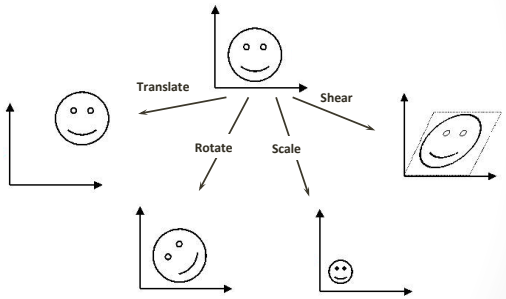
2D Modeling Transformations



2D Modeling Transformations

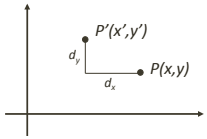


2D Geometrical Transformations



Translate Points

- We can translate points in the (*x, y*) plane to new positions by adding translation amounts to the coordinates of the points. For each point *P*(*x, y*) to be moved by d_x units parallel to the *x* axis and by d_y units parallel to the *y* axis, to the new point *P'*(*x', y'*). The translation has the following form:



$$x' = x + d_x$$
$$y' = y + d_y$$

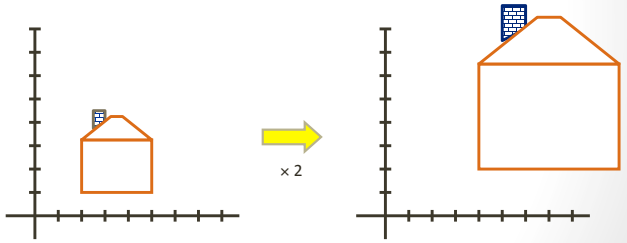
In matrix format:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

If we define the translation matrix $T = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$ then we have *P' =P + T.*

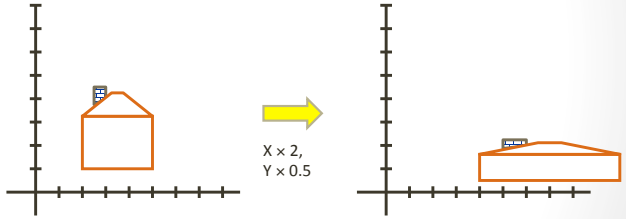
scaling

- Scaling a coordinate means multiplying each of its components by a scalar
- Uniform scaling means this scalar is the same for all components:



scaling

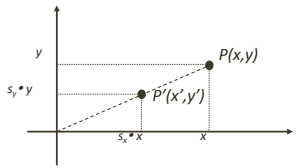
- Non-uniform scaling: different scalars per component:



- How can we represent this in matrix form?

Scale points

- Points can be scaled (stretched) by sx along the x axis and by sy along the y axis into the new points by the multiplications:
- We can specify how much bigger or smaller by means of a "scale factor"
- To double the size of an object we use a scale factor of 2, to half the size of an object we use a scale factor of 0.5

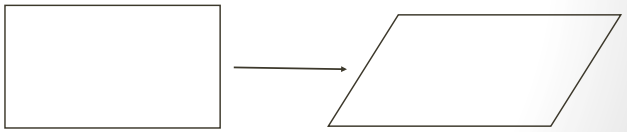


$$x' = s_x x$$
$$y' = s_y y$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

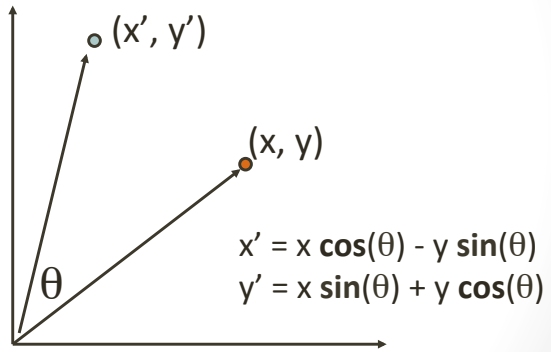
If we define $S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$ then we have *P' =SP*

shearing

$$Shear = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \qquad S^{-1} = \begin{pmatrix} 1 & -a \\ 0 & 1 \end{pmatrix}$$



2-D Rotation



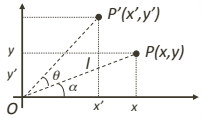
2-D Rotation

Points can be rotated through an angle θ about the origin:

$|OP'| = |OP| = l$

$$x' = |OP'|\cos(\alpha+\theta) = l\cos(\alpha+\theta)$$
$$= l\cos\alpha\cos\theta - l\sin\alpha\sin\theta$$
$$= x\cos\theta - y\sin\theta$$

$$y' = |OP'|\sin(\alpha+\theta) = l\sin(\alpha+\theta)$$
$$= l\cos\alpha\sin\theta + l\sin\alpha\cos\theta$$
$$= x\sin\theta + y\cos\theta$$



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$P' = RP$

review

- Translate: P' = P+T
- Scale: P' = SP
- Rotate: P' = RP

- Spot the odd one out…
  - Multiplying versus adding matrix…
  - Ideally, all transformations would be the same..
    - easier to code
- Solution: Homogeneous Coordinates

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Identity?

$$x' = x \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = y$$

2D Scale around (0,0)?

$$x' = s_x * x \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = s_y * y$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

$$x' = \cos\Theta * x - \sin\Theta * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

2D Shear?

$$x' = x + sh_x * y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = sh_y * x + y$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Mirror about Y axis?

$$x' = -x \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = y$$

2D Mirror over (0,0)?

$$x' = -x \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$y' = -y$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$x' = x + t_x \quad \text{NO!}$$
$$y' = y + t_y$$

Only linear 2D transformations can be represented with a 2x2 matrix

Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved

Homogeneous Coordinates

For a given 2D coordinates (*x, y*), we introduce a third dimension:

[*x, y, 1*]

In general, a homogeneous coordinates for a 2D point has the form:

[*x, y, W*]

Two homogeneous coordinates [*x, y, W*] and [*x', y', W'*] are said to be of the same (or equivalent) if

x = kx'
y = ky' for some $k \neq 0$
W = kW'

eg: [2, 3, 6] = [4, 6, 12] where k=2

Therefore any [*x, y, W*] can be normalised by dividing each element by *W*:
[*x/W, y/W*, 1]

Homogeneous Coordinates

- Homogeneous coordinates
  - represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{\text{homogeneous coords}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous coordinates seem unintuitive, but they make graphics operations much easier

Homogeneous Coordinates

- Q: How can we represent translation as a 3x3 matrix?

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}_x$$
$$\boldsymbol{y}' = \boldsymbol{y} + \boldsymbol{t}_y$$

Homogeneous Coordinates

- Q: How can we represent translation as a 3x3 matrix?

$$x' = x + t_x$$
$$y' = y + t_y$$

- A: Using the rightmost column:

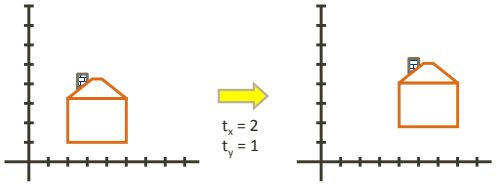
$$\mathbf{Translation} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Translation

- Example of translation

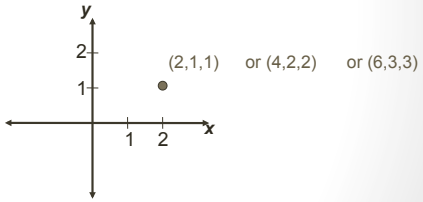
Homogeneous Coordinates

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



Homogeneous Coordinates

- Add a 3rd coordinate to every 2D point
  - (x, y, w) represents a point at location (x/w, y/w)
  - (x, y, 0) represents a point at infinity
  - (0, 0, 0) is not allowed



Convenient coordinate system to represent many useful transformations

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

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

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
Rotate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
Shear

Rigid-Body vs. Affine Transformations

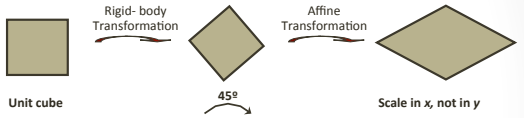
- A transformation matrix of the form

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

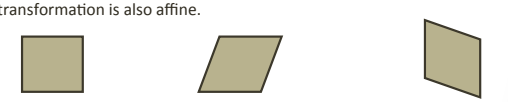
- where the upper 2×2 sub-matrix is orthogonal, preserves angles and lengths. Such transforms are called **rigid-body** transformations, because the body or object being transformed is not distorted in any way. An arbitrary sequence of rotation and translation matrices creates a matrix of this form.

- The product of an arbitrary sequence of rotation, translations, ***and scale*** matrices will cause an **affine** transformation, which have the property of preserving parallelism of lines, but not of lengths and angles.

Rigid-Body vs. Affine Transformations



Shear transformation is also affine.



$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of 2D Transformations

1. Additivity of successive translations

We want to translate a point P to P' by $T(d_{x1}, d_{y1})$ and then to P'' by another $T(d_{x2}, d_{y2})$

$$P'' = T(d_{x2}, d_{y2})P' = T(d_{x2}, d_{y2})[T(d_{x1}, d_{y1})P]$$

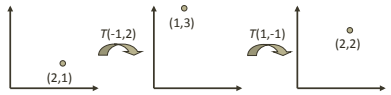
On the other hand, we can define T_{21} first, then apply T_{21} to P:

$$T_{21} = T(d_{x2}, d_{y2})T(d_{x1}, d_{y1}) = \begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1}+d_{x2} \\ 0 & 1 & d_{y1}+d_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\boxed{P'' = T_{21}P}$$

Examples of Composite 2D Transformations



$$T_{21} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of 2D Transformations

2. Multiplicativity of successive scalings

$$\begin{aligned} P'' &= S(s_{x2}, s_{y2})[S(s_{x1}, s_{y1})P] \\ &= [S(s_{x2}, s_{y2})S(s_{x1}, s_{y1})]P \\ &= S_{21}P \end{aligned}$$

where $S_{21} = S(s_{x2}, s_{y2})S(s_{x1}, s_{y1})$

$$= \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_{x2} * s_{x1} & 0 & 0 \\ 0 & s_{y2} * s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of 2D Transformations

3. Additivity of successive rotations

$$\begin{aligned} P'' &= R(\theta_2)[R(\theta_1)P] \\ &= [R(\theta_2)R(\theta_1)]P \\ &= R_{21}P \end{aligned}$$

where $R_{21} = R(\theta_2)R(\theta_1)$

$$= \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_2 + \theta_1) & -\sin(\theta_2 + \theta_1) & 0 \\ \sin(\theta_2 + \theta_1) & \cos(\theta_2 + \theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of 2D Transformations

4. Different types of elementary transformations discussed above can be concatenated as well.

$$\begin{aligned} P' &= R(\theta)[T(d_x, d_y)P] \\ &= [R(\theta)T(d_x, d_y)]P \\ &= MP \end{aligned}$$

where $M = R(\theta)T(d_x, d_y)$

Order matters!

- **As we said, the order for composition of 2D geometrical transformations matters, because, in general, matrix multiplication is not commutative. However, it is easy to show that, in the following four cases, commutativity holds:**
  1. **Translation + Translation**
  2. **Scaling + Scaling**
  3. **Rotation + Rotation**
  4. **Scaling (with $s_x = s_y$) + Rotation**
- **just to verify case 4: if $s_x = s_y$, $M_1 = M_2$.**

$$M_1 = S(s_x, s_y)R(\theta)$$

$$= \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x * \cos\theta & -s_x * \sin\theta & 0 \\ s_y * \sin\theta & s_y * \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_2 = R(\theta)S(s_x, s_y)$$

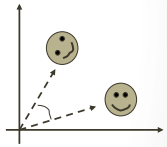
$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x * \cos\theta & -s_y * \sin\theta & 0 \\ s_x * \sin\theta & s_y * \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation Revisit

- The standard rotation matrix is used to rotate about the origin (0,0)

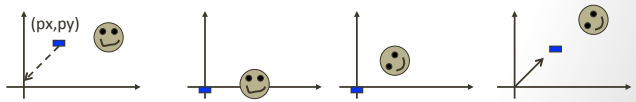
$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- What if I want to rotate about an arbitrary center?

Arbitrary Rotation Center

- To rotate about an arbitrary point P (px,py) by θ:
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Rotate the object: R(θ)
  - Translate the object back: T(px,py)



Arbitrary Rotation Center

- Translate the object so that P will coincide with the origin: T(-px, -py)
- Rotate the object: R(θ)
- Translate the object back: T(px,py)

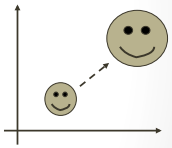
- Put in matrix form: T(px,py) R(θ) T(-px, -py) * P

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & px \\ 0 & 1 & py \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -px \\ 0 & 1 & -py \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Scaling Revisit

- The standard scaling matrix will only anchor at (0,0)

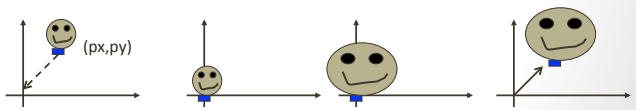
$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- What if I want to scale about an arbitrary pivot point?

Arbitrary Scaling Pivot

- To scale about an arbitrary pivot point P (px,py):
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Scale the object: S(sx, sy)
  - Translate the object back: T(px,py)



Composing Transformation

- Composing Transformation – the process of applying several transformation in succession to form one overall transformation
- If we apply transform a point P using M1 matrix first, and then transform using M2, and then M3, then we have:

(M3 x (M2 x (M1 x P))) = M3 x M2 x M1 x P

(pre-multiply)

M

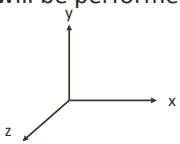
In Conclusion

- Matrix multiplication is associative
  M3 x M2 x M1 = (M3 x M2) x M1 = M3 x (M2 x M1)
- Transformation products may not be commutative A x B != B x A
- Some cases where A x B = B x A

A	B
translation	translation
scaling	scaling
rotation	rotation
uniform scaling (sx = sy)	rotation

Three-Dimensional Graphics

- A 3D point (x,y,z) – x,y, and Z coordinates
- We will still use column vectors to represent ^{T} points
- Homogeneous coordinates of a 3D point (x,y,z,1)
- Transformation will be performed using 4x4 matrix



Homogeneous coordinates

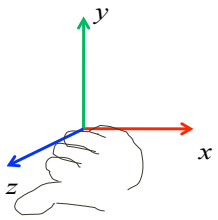
- Homogeneous coordinates (review)
  - 4X4 matrix used to represent translation, scaling, and rotation
  - a point in the space is represented as

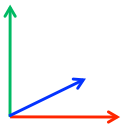
$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - Treat all transformations the same so that they can be easily combined

Right-handed Coordinate System

$$x \times y = z;\ \ y \times z = x;\ \ z \times x = y$$





Left hand coordinate system
Not used in this class and
Not in OpenGL

3D Transformation

- Very similar to 2D transformation
- Translation transformation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_x \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Homogenous
coordinates

3D Transformation

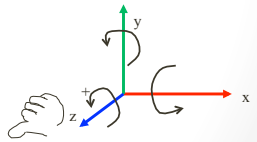
- Very similar to 2D transformation
- Scaling transformation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Homogenous coordinates

3D Rotation

- 3D rotation is done around a rotation **axis**
- Fundamental rotations – rotate about x, y, or z axes
- Counter-clockwise rotation is referred to as positive rotation (when you look down negative axis)

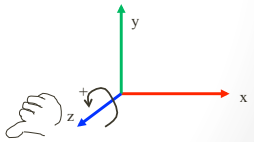


3D Rotation

- Rotation about z – similar to 2D rotation

$$x' = x\cos\alpha - y\sin\alpha$$
$$y' = x\sin\alpha + y\cos\alpha$$
$$z' = z$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

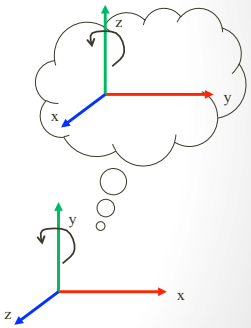


3D Rotation

- Rotation about y: z -> y, y -> x, x->z

$$z' = z\cos\beta - x\sin\beta$$
$$x' = z\sin\beta + x\cos\beta$$
$$y' = y$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

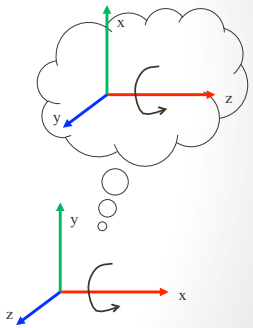


3D Rotation

- Rotation about x (z -> x, y -> z, x->y)

$$y' = y\cos\gamma - z\sin\gamma$$
$$z' = y\sin\gamma + z\cos\gamma$$
$$x' = x$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma & 0 \\ 0 & \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ X axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ Y axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ Z axis

New points rotation matrix old points

Euler Angles

- This means that we can represent an orientation with 3 numbers
- A sequence of rotations around principle axes is called an *Euler Angle Sequence*
- Assuming we limit ourselves to 3 rotations without successive rotations about the same axis, we could use any of the following 12 sequences:

XYZ	XZY	XYX	XZX
YXZ	YZX	YXY	YZY
ZXY	ZYX	ZXZ	ZYZ

Euler Angles

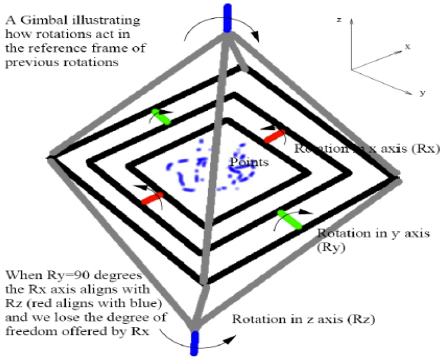
- This gives us 12 redundant ways to store an orientation using Euler angles
- Different industries use different conventions for handling Euler angles (or no conventions)
- To use Euler angles, one must choose which of the 12 representations they want
- There may be some practical differences between them and the best sequence may depend on what exactly you are trying to accomplish

Gimbal Lock

- A Gimbal is a hardware implementation of Euler angles used for mounting gyroscopes or expensive globes

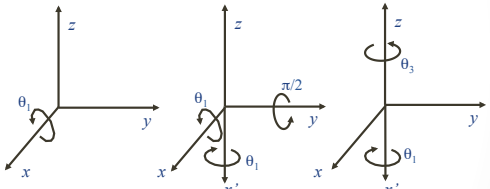
- Gimbal lock is a basic problem with representing 3D rotation using Euler angles

Gimbal Lock



Gimbal Lock

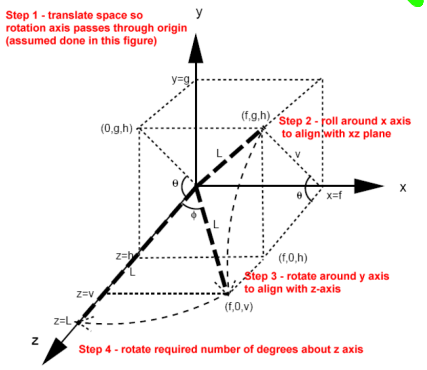
- A 90 degree rotation about the y axis essentially makes the first axis of rotation align with the third.



- Incremental changes in x,z produce the same results – you've lost a degree of freedom

Arbitrary rotation axis

7 matrices needed
Step1 2 3 4 3 2 1



Multimedia Systems

– 3D Multimedia

Dr. Yi-Zhe Song

EBU706U

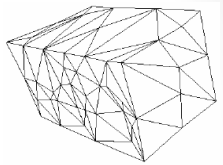
Some definitions (reminder)

- Modelling
  - Representing 3D objects (as 3D models)
- Imaging
  - Representing 2D images
- Rendering
  - Constructing 2D images from 3D models

EBU706U

Modelling and Rendering

- Modelling
  - polygonal meshes
  - splines and subdivision surfaces
  - volumetric representations
- Rendering
  - lighting models
  - global illumination
  - image-based rendering



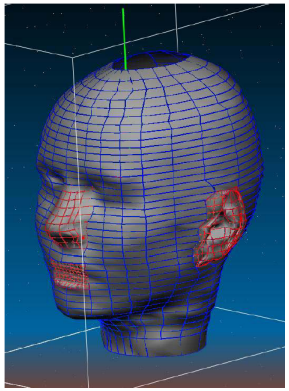
EBU706U

Today's agenda

- Model-based 3D (reminder)
- Camera model (reminder)
- Image-based rendering

EBU706U

Model-based 3D



EBU706U

Digitising devices

facial/object scanner

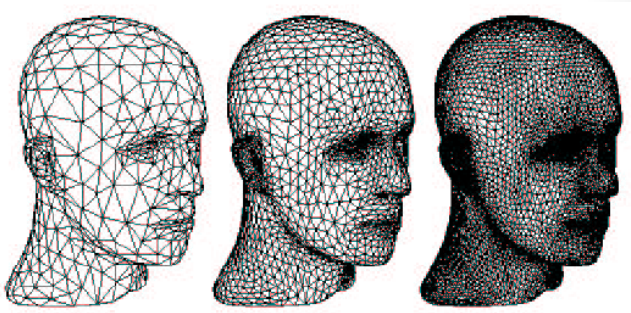
dental scanner

EBU706U

Digitising devices

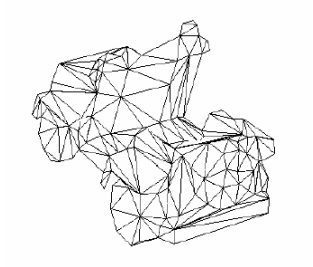
EBU706U

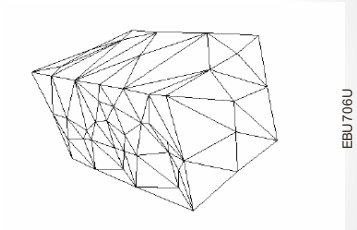
Wireframe model



EBU706U

Wireframe

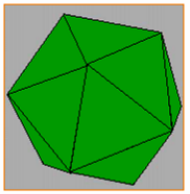




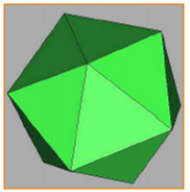
EBU706U

Illumination

- Direct illumination
  - emission at light source
  - scattering at surface
- Global illumination
  - shadows
  - inter-object reflections
  - refractions



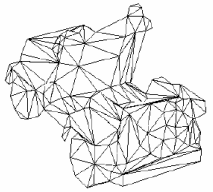
without illumination

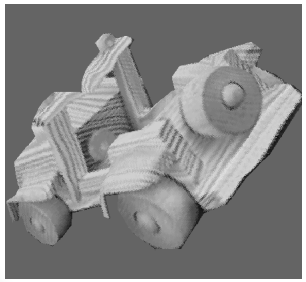


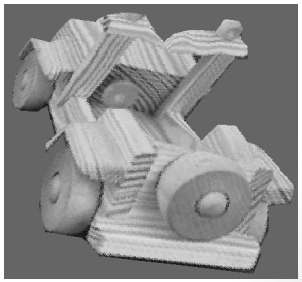
with illumination

EBU706U

Rendering







EBU706U

Texture mapping

EBU706U

Applications

- 3D game software development
  - quickly and easily scan and digitise character models
- Inspection
  - quality assurance
  - production line surface measurement
- Medical, surgical & dental
  - orthopaedic
  - prosthetic
  - maxillofacial
- Industrial design
  - rapid prototyping
  - reverse engineering
- Very low bit-rate coding

EBU706U

Applications

- Animation and virtual reality
  - create 3D characters and environments for TV and movies
- Education
  - training, simulation, and digitising design models
- Architecture
  - design work at mock-up level
- Fashion and textiles
  - fitting clothes and determining dimensions
- Museums
  - 3D archiving
  - cataloguing
  - taking inventory of collections

EBU706U

Today's agenda

- Model-based 3D
- Camera model
- Image-based rendering (IBR)

EBU706U

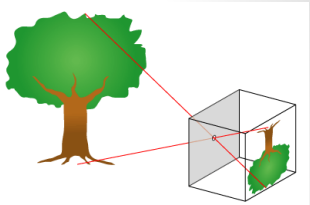
Image formation

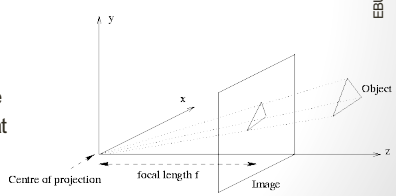
- Human vision:
  - The process of using light reflected from the surrounding world as a way of interpreting or understanding it
  - Understanding the 3D world from the 2D projection on the retina of the eye
- Computer vision:
  - Human vision can be modelled (to some extent) and implemented in a computer using cameras and image interpretation, e.g.
    - edge extraction
    - feature extraction
    - feature analysis
    - pattern recognition

EBU706U

Camera model

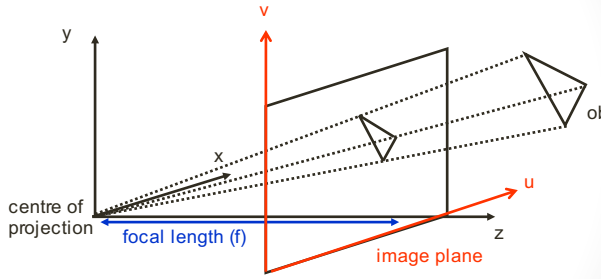
- Pinhole camera model
  - simplest and ideal model of camera function
  - has an infinitesimally small hole through which light enters before forming an *inverted image* on the camera surface facing the hole
- Simplified model
  - The image plane is placed between the focal point of the camera and the object, so that the image is not inverted





EBU706

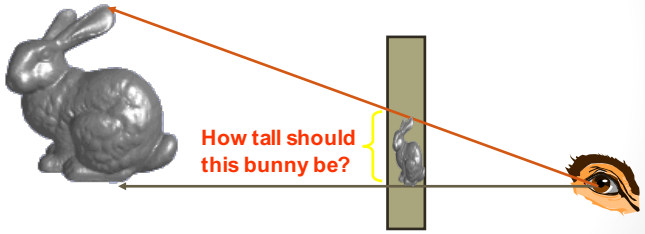
Camera model



EBU706U

Perspective projection

- 3-D graphics → think of the screen as a 2-D window onto the 3-D world

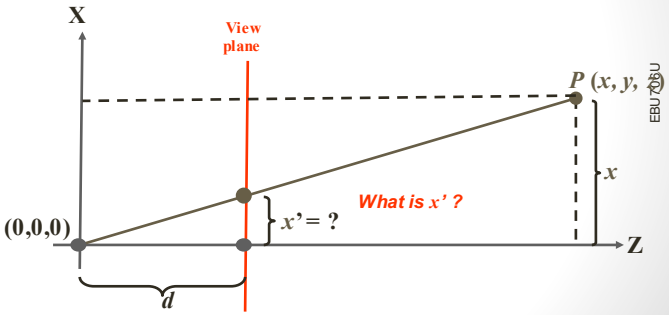


EBU706U

Perspective projection

- The geometry of the situation is that of similar triangles.

View from above

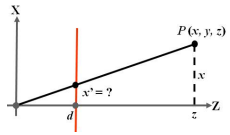


EBU706U

Perspective projection

- Desired result for a point $[x, y, z, 1]^T$ projected onto the view plane:

$$\frac{x'}{d} = \frac{x}{z} \quad \frac{y'}{d} = \frac{y}{z}$$



$$x' = \frac{d \cdot x}{z} = \frac{x}{z/d} \quad y' = \frac{d \cdot y}{z} = \frac{y}{z/d} \quad z = d$$

What could a matrix look like to do this?

EBU706U

Perspective projection matrix

$$\begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- in 3-D coordinates: $\left(\frac{x}{z/d}, \frac{y}{z/d}, d \right)$

EBU706U

Perspective geometry

- Perspective projection
  - projection of a 3D object onto a 2D surface by straight lines that pass through a single point
- If we denote the distance of the image plane to the centre of projection by *d*
  - then the image coordinates *(u,v)* are related to the object coordinates *(x, y, z)* by
    - *u = dx/z*
    - *v = dy/z*

EBU706U

Today's agenda

- Model-based 3D
- Camera model
- Image-based rendering (IBR)

EBU706U

Image-based rendering

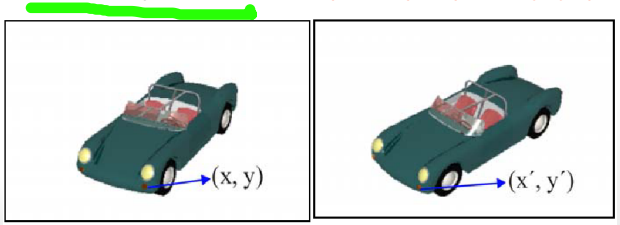
- An approach to rendering in which objects and environments are modelled using image data instead of geometric primitives ..
- Computer vision is mostly focused on detecting, grouping, and extracting features (edges, faces, *etc.*) present in a given picture and then trying to interpret them as three-dimensional clues.
- Image-based modelling and rendering use multiple two-dimensional images in order to generate directly novel two-dimensional images, skipping the manual modelling stage.

EBU706U

Correspondence problem

- To find the position of a pixel representing the same real world point projected into two slightly different image planes
- Once the correspondence problem has been solved, resulting in a set of image points which are in correspondence, other methods can be applied to this set to reconstruct the position, motion and/or rotation of the corresponding 3D points in the scene.

Disparity Vector : $D = (d_1, d_2) = (x-x', y-y')$



EBU706U

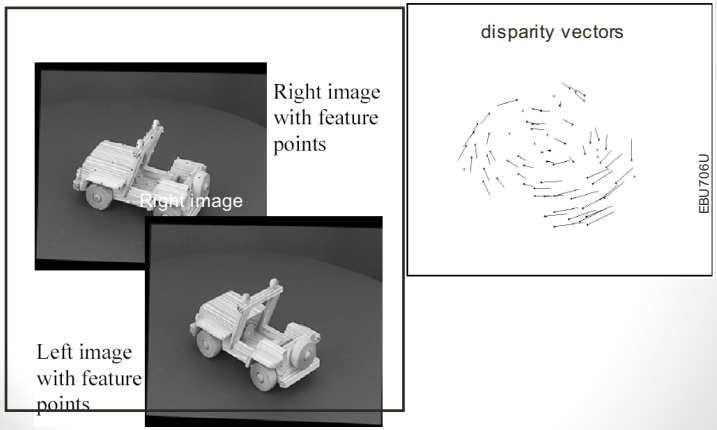
problem: cannot find correspondence between pixels located in a uniformly–coloured region

Estimating correspondence

- using feature points
- or matching macro blocks

EBU706U

Disparity estimation: example



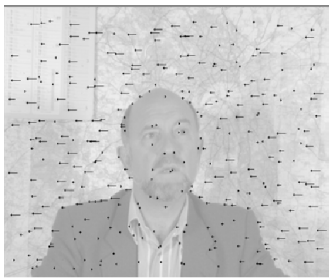
EBU706U

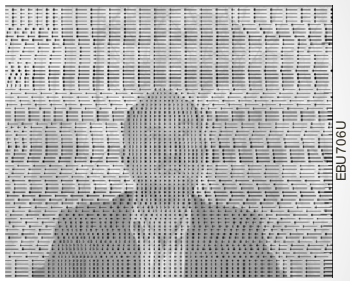
Image-based rendering (IBR)

- IBR: simple approach
  - Solve the correspondence problem for *well selected* image points
  - Generate dense disparity maps by interpolation
  - Generate intermediate views by direct interpolation using disparity information

EBU706U

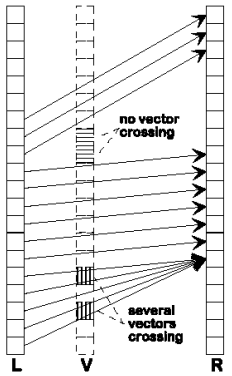
Disparity estimation and interpolation

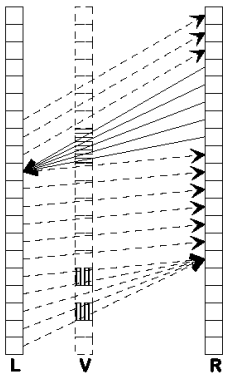




EBU706U

Projection on intermediate image plane





EBU706U

left

right

Sparse disparity field

left

right

Virtual view generation

What did we learn today?

- Model-based 3D
- Camera model
- Image-based rendering (IBR)

EBU706U

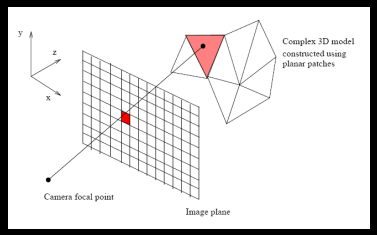
Question

Explain the difference between geometry-based rendering and image-based rendering.

EBU706U

Image Formation

- In OpenGL we model using planar patches
  - Convex polygons – usually triangles



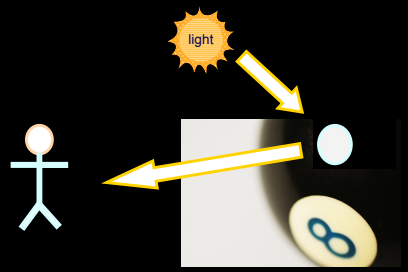
- What colour are the pixels in the rendered image?
  - Simple case – just the colour of the surface
  - Usual case – take lighting into account for shading

Phong Illumination Model

- Emulating real-world lighting is complex
  - Full formulation is in Kajya's lighting eqn. (Adv. Gfx)
  - A **fast** and usually **adequate** approximation is PIM
- PIM – Light consists of 3 additive components
  - Specular I_s
  - Diffuse I_d
  - Ambient I_a
- Colour of pixel = $I_s + I_d + I_a$
  - So we need to work out each of the 3 contributions to compute the colour of the pixel...

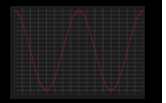
Phong Illumination Model

- Specular component
  - Shiny
  - Varies according to viewpoint
  - Is the colour of the incident light



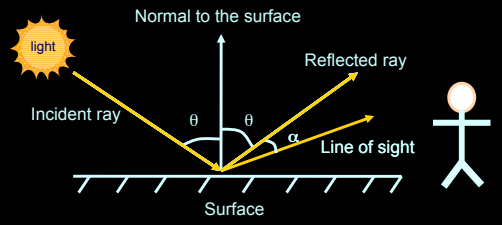
Specular Component

- How does Phong model it?



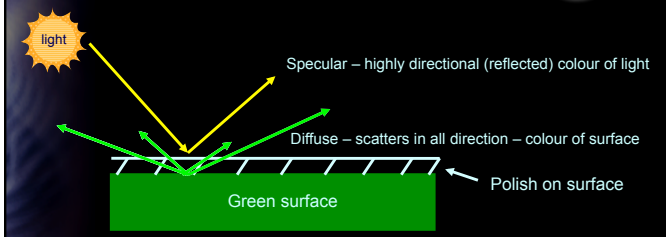
$$I_s = I_p \quad k \quad \cos^n \alpha$$

Specular — Light source — Dampens according to α



Diffuse Component

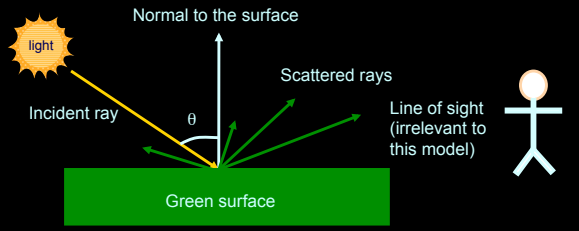
- Imagine model surface as 2 layers
  - Polish on the surface
  - Matte surface underneath



- The matte surface is green because it reflects the green parts of light
- The light reaching the diffuse layer (matte surface) is the light emitted from the light source *minus* the light reflected by the specular (polish) layer

Diffuse Component

- How does Phong model it?
  - Using Lambert's Law
    - The light reflected from a matte surface is uniformly scattered
    - The amount of light (intensity) is proportional to the angle between the surface normal and the light source



Diffuse Component

- How does Phong model it?

$$I_d = (I_p - I_s) \quad k \quad \cos\theta$$

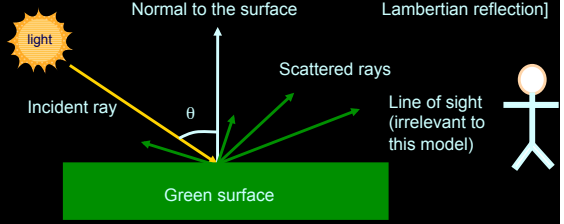
Specular

Light reaching diffuse layer

How much of this colour does surface reflect?

Dampens according to θ

[Lambert's Law = Lambertian reflection]



Phong Illumination Model

- Compute pixels as specular + diffuse results in high contrast images that do not look real
- Environments have a certain amount of complex, scattered light that enters our eye too
  - This is **Global Illumination** / ***ambient light***
  - We model this as a small constant amount of light which we add to all pixels as I_a

Final colour of pixel = $I_a + I_d + I_s$

We compute the Phong calculation three times, for Red Green and Blue

Phong Illumination Model

- To summarise visually...

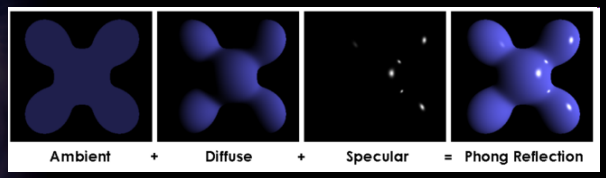


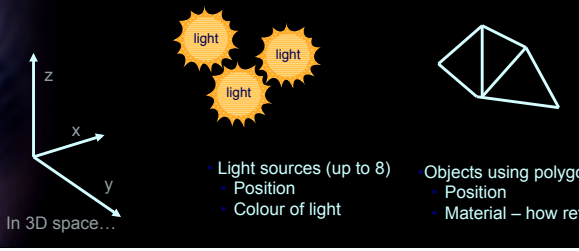
Image © Wikimedia CCL

Final colour of pixel = $I_a + I_d + I_s$

We compute the Phong calculation three times, for Red Green and Blue

OpenGL Lighting

- OpenGL Implements the Phong Illumination Model
  - A slightly more general version
- To model a scene in OpenGL we specify...



- Light sources (up to 8)
  - Position
  - Colour of light
- Objects using polygons
  - Position
  - Material – how reflect light

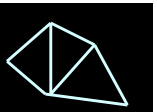
OpenGL Objects

- Specifying Polygons and the materials they are made from
  - First we look at polygon position
  - Polygons must be planar and convex (triangles safe = always are)

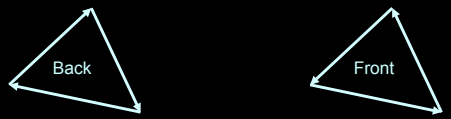
```
glBegin(GL_TRIANGLES);

        glNormal3f(x,y,z); // you must manually specify the normals
        glVertex3f(1,3,2);
        glVertex3f(4,5,6);
        glVertex3f(3,2,3);      } Models a triangular surface patch

        glNormal3f(x,y,z);
        glVertex3f(x,y,z);
        glVertex3f(x,y,z);
        glVertex3f(x,y,z);      } Another patch

        etc

glEnd();
```

- GL_QUADS - Four glVertex3f calls per patch for the 4 corners
- GL_POLYGONS - Pass array of vertices to glVertexf

OpenGL Objects



- The concept of a polygon's front side and back side
  - Depends on the **winding** of the vertices



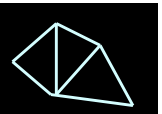
Clockwise

Anti-Clockwise
(Counter-Clockwise = CCW)

- You can change OpenGL's definition of front and back
  - Usually inadvisable as " everyone knows" convention Front = CCW

```
glFrontFace(GL_CCW); // default - front is anticlockwise
glFrontFace(GL_CW);  // front is now clockwise
```

OpenGL Objects



- Specifying Polygons and the materials they are made from
  - What are polygons made from? **Material** that reflects light

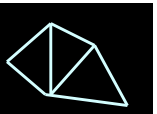
```
GLfloat spec[] = {1, 1, 1, 1};
GLfloat diff[] = {0, 1, 0, 1};
GLfloat amb[]  = {0, 1, 0, 1};

glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, spec);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diff);
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, amb);


glBegin(GL_TRIANGLES);

        glVertex3f(1,3,2);
        etc

glEnd();
```

Green shiny material

Models a patch in current material i.e. green and shiny

- Wouldn't it be tedious if we needed to define a new material in this way, every time we changed colour in our model?

OpenGL Objects



- Specifying colour Materials the easy way = **colour tracking**

```
GLfloat spec[] = {1, 1, 1, 1};

glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, spec);
glMaterialfi (GL_FRONT_AND_BACK, GL_SHININESS, 96);

glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);

glBegin(GL_TRIANGLES);

        glColor3f(0,1,0); // Green
        glVertex3f(1,3,2);  x3

        glColor3f(1,0,0); // Red
        glVertex3f(1,3,2);  x3

glEnd();
```

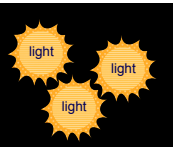
setup shiny material

enable 'colour tracking' on diffuse and ambient components of material

Each time we change colour, the DIFFUSE And AMBIENT parts of the material get changed but SPECULAR parts stay as originally defined

- In OpenGL we almost always set up materials using colour tracking

OpenGL Lighting



- Specifying Light Sources
  - Inside the display callback i.e. run when drawing the graphics

```
GLfloat light_position[] = { x, y, z, 0.0 };

GLfloat white[]       = { 1, 1, 1, 1 };
GLfloat red[]         = { 1, 0, 0, 1 };
GLfloat anycolour[]   = { red, green, blue, 1 };

glEnable(GL_LIGHTING);

glEnable(GL_LIGHT0);

glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_AMBIENT, red);
glLightfv(GL_LIGHT0, GL_SPECULAR, white);
```

OpenGL Lighting

- Specifying Light Sources
  - First, set up the ambient light everywhere in the scene

```
GLfloat dim_white[]   = { 0.2, 0.2, 0.2, 1 };
GLfloat red[]         = { 1, 0, 0, 1 };
GLfloat anycolour[]   = { red, green, blue, 1 };

glEnable(GL_LIGHTING);

glLightModel(GL_LIGHT_MODEL_AMBIENT, dim_white);
```

- If you don't do this, the default behaviour of OpenGL is to introduce 20% intensity white light just as in the above code.
  - Allows coders to see where their objects are before writing any detailed lighting code

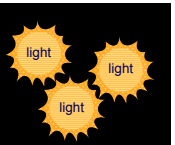
OpenGL Lighting

- Directional light sources
  - As described so far, lights are omni-directional
  - But we can create directed light sources (spotlights)

```
glEnable(GL_LIGHTING);

glEnable(GL_LIGHT0);

glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_AMBIENT, red);
glLightfv(GL_LIGHT0, GL_SPECULAR, white);

GLfloat light_dir[] = { x, y, z, 0.0 };   // unit vector

glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light_dir);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 1000);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 1.0);

…
```

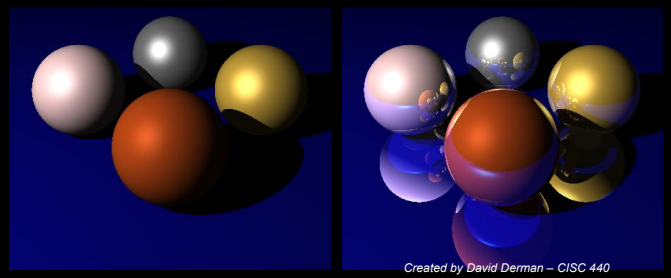
} make into spotlight

OpenGL Lighting



- This is not the end of the lighting story!
  - But it is, for this course.
- In the Phong Model all lights are point sources
  - But in the world we have panels of light e.g. strip lighting
  - OpenGL has a fourth lighting component called GL_EMISSIVE that lets you create glowing surfaces
  - But the Phong model still does not properly handle non-point sources; it does not handle reflected light between surfaces.
  - Radiosity rendering and the BDRF solve this – covered on the Advanved Gfx. Course

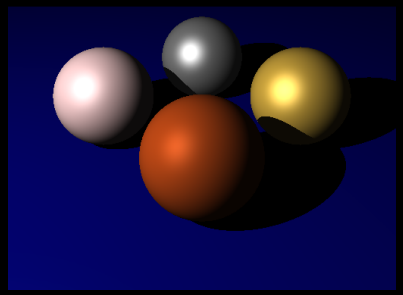
Photo-Realism



Created by David Derman – CISC 440

1

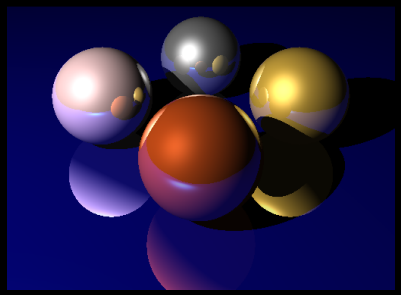
Photo-Realism



Created by David Derman – CISC 440

2

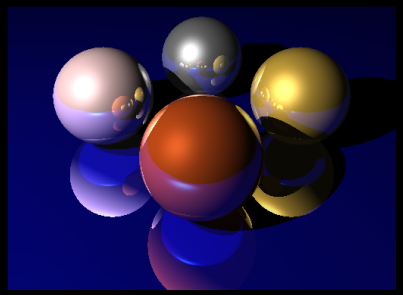
Photo-Realism



Created by David Derman – CISC 440

3

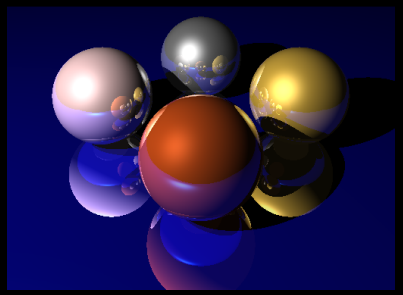
Photo-Realism



Created by David Derman – CISC 440

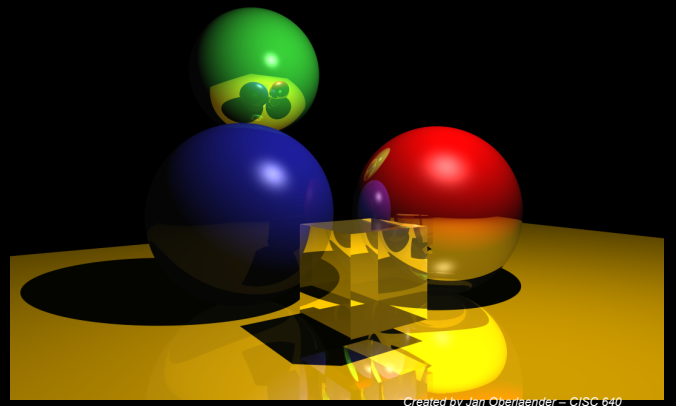
4

Photo-Realism



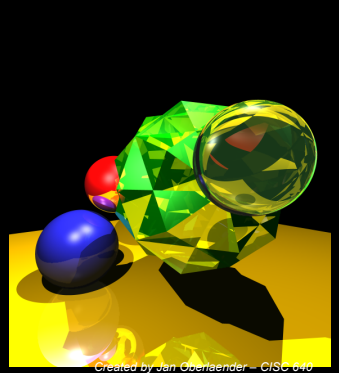
Created by David Derman – CISC 440

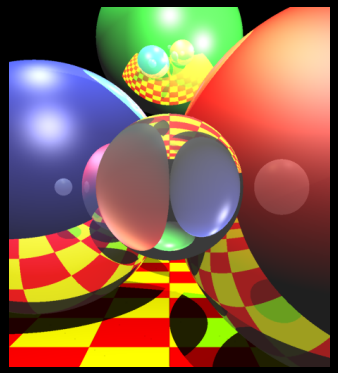
5



Created by Jan Oberlaender – CISC 640

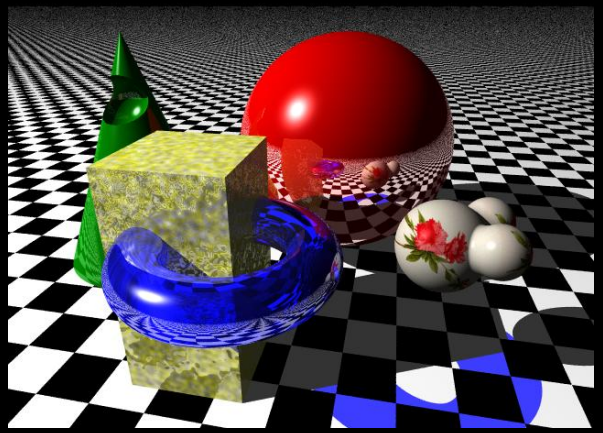
6





Created by Jan Oberlaender – CISC 640

7



Created by Donald Hyatt
http://www.tjhsst.edu/~dhyatt/superap/povray.html

8

Introduction - Light

- Three Ideas about light
  - Light rays travel in straight lines
  - Light rays do not interfere with each other if they cross
  - Light rays travel from light source to the eye, but the physics is invariant under path reversal (Helmholtz reciprocity)
    - P. Sen, et al., "Dual Photography", SIGGRAPH 2005
      - Novel photographic technique to interchange the lights and cameras in a scene

9

Introduction - Ray tracing

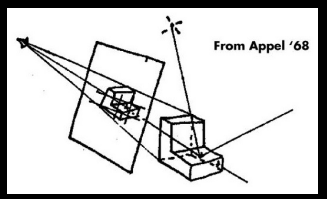
- What is Ray Tracing?
  - Ray Tracing is a global illumination based rendering method for generating realistic images on the computer
- Originators
  - Appel 1968
  - Goldstein and Nagel
  - Whited 1979

History	
Polygons	Appel '68
Quadrics, CSG	Goldstein and Nagel '71
Tori	Roth '82
Bicubic patches	Whitted '80, Kajiya '82
Superquadrics	Edwards and Barr '83
Algebraic surfaces	Hanrahan '82
Swept surfaces	Kajiya '83, van Wijk '84
Fractals	Kajiya '83
Height fields	Coquillart and Gangnet '84
Deformations	Barr '86

Courtesy of Pat Hanrahan, Computer Graphics: Image Synthesis Techniques

10

Introduction – Ray tracing

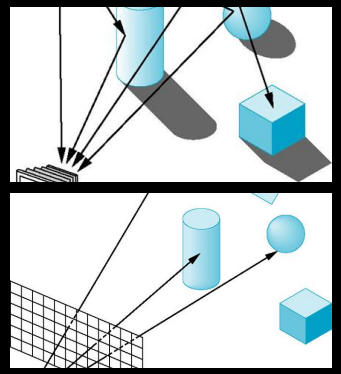


- Appel
  - Ray Casting
- Goldstein and Nagel
  - Scene Illumination
- Whited
  - Recursive ray tracing (reflection and refraction)
  - Forward and Backward Ray tracing

Courtesy of Pat Hanrahan, Computer Graphics: Image Synthesis Techniques

11

Introduction – Ray tracing

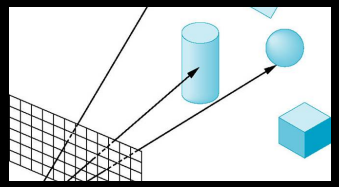


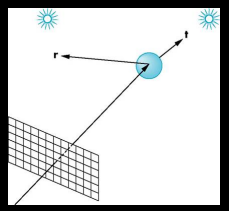
- Forward Ray tracing
  - Rays from light source bounce of objects before reaching the camera
  - Computational wastage
- Backward Ray tracing
  - Track only those rays that finally made it to the camera

Courtesy: Angel

12

Ray Casting





- Ray Casting
  - visible surfaces of objects are found by throwing (or casting) rays of light from the viewer into the scene
- Ray Tracing
  - Extension to Ray casting
  - Recursively cast rays from the points of intersection

Courtesy: Angel

13

Ray casting

- In ray casting, a ray of light is traced in a backwards direction.
  - We start from the eye or camera and trace the ray through a pixel in the image plane into the scene and determine what it intersects
  - The pixel is then set to the color values returned by the ray.
  - If the ray misses all objects, then that pixel is shaded the background color

14

Algorithm – Ray casting

```
 objects and light sources in the scene
 camera
  r = 0; r < nRows; r++)
nt c = 0; c < nCols; c++)

ld the rc-th ray
d all intersections of the rc-th ray with objects in the scene
ntify the intersection that lies closest to, and in front of, the eye
mpute the "hit point" where the ray hits this object, and the normal vector at tha
d the color of the light returning to the eye along the ray from the point of inter
ce the color in the rc-th pixel.
```

15

Ray tracing - overview

- Ray casting finds the visible surfaces of objects
- Ray tracing determines what each visible surface looks like
  - This extra curiosity is quite heavy on your processor
- But it allows you to create effects that are very difficult or even impossible to do using other methods.
  - Reflection
  - Transparency
  - Shadows

Courtesy: http://www.geocities.com/jamisbuck/raytracing.html

16

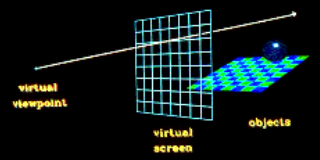
Ray tracing - overview

- Ray tracing algorithm is a "finitely" recursive image rendering
- First stage: Like ray casting
  - Shoot a ray from the eye (Primary ray)
  - Determine all the objects that intersect the ray
  - Find the nearest of intersections
- Second stage:
  - Recurses by shooting more rays from the point of intersection (Secondary rays)
    - Find the objects that are reflected at that point
    - Find other objects that may be seen through the object at that point
    - Find out light sources that are directly visible from that point
- Repeat the second stage until required

17

Ray Tracing – example

- Sometimes a ray misses all objects

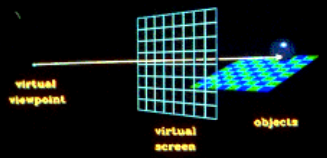


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

18

Ray Tracing – example

- Sometimes a ray hits an object

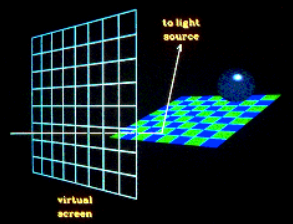


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

19

Ray Tracing – example

- Is the intersected point in shadow?
  - "Shadow Rays" to light source

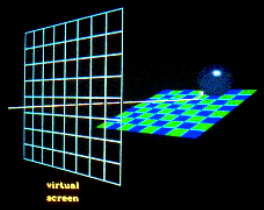


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

20

Ray Tracing – example

- Shadow rays intersect another object
  - First intersection point in shadow of the second object

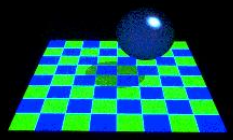


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

21

Ray Tracing – example

- Shadow rays intersect another object
  - First intersection point in shadow of the second object

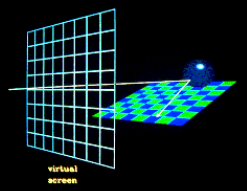


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

22

Ray Tracing - example

- Reflected ray generated at point of intersection
  - Tested with all the objects in the scene

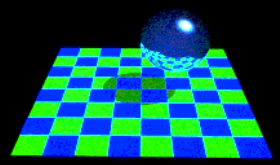


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

23

Ray Tracing - example

- Local illumination model applied at the point of intersection

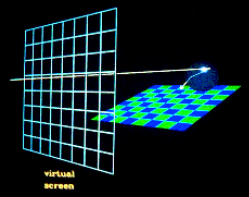


created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

24

Ray Tracing – example

- Transparent object
  - Spawn a transmitted ray and test against all objects in the scene



created by Michael Sweeny, et al for ACM SIGGRAPH Education slide set 1991

25

Requirements for Ray tracing

- Compute 3D ray into the scene for each 2D image pixel
- Compute 3D intersection point of ray with nearest object in scene
  - Test each primitive in the scene for intersection
  - Find nearest intersection
- Recursively spawn rays from the point of intersection
  - Shadow Rays
  - Reflected rays
  - Transmitted rays
- Accumulate the color from each of the spawned rays at the point of intersection

26

Ray object intersection

- Equation of a ray $r(t) = \mathbf{S} + \mathbf{c}t$
  - ***"S"*** is the starting point and **"c"** is the direction of the ray
- Given a surface in implicit form *F(x,y,z)*
  - *plane:* $F(x,y,z) = ax + by + cz + d = \mathbf{n} \cdot \mathbf{x} + d$
  - *sphere:* $F(x,y,z) = x^2 + y^2 + z^2 - 1$
  - *cylinder:* $F(x,y,z) = x^2 + y^2 - 1 \qquad 0 < z < 1$
- All points on the surface satisfy *F(x,y,z)=0*
- Thus for ray *r(t)* to intersect the surface $F(r(t)) = 0$
- The hit time can be got by solving $F(\mathbf{S} + \mathbf{c}t_{hit}) = 0$

27

Ray plane intersection

- Equation of a ray $r(t) = \mathbf{S} + \mathbf{c}t$
- Equation of a plane $F(x,y,z) = ax + by + cz + d = \mathbf{n} \cdot \mathbf{x} + d$
  - **n** is the normal to the plane and d is the distance of the plane from the origin
- Substituting and solving for t

$$F(x,y,z)|_{r(t)} = \mathbf{n} \cdot \mathbf{x} + d|_{r(t)} = 0$$
$$= \mathbf{n} \cdot (\mathbf{S} + \mathbf{c}t) + d = 0$$

$$\mathbf{n} \cdot \mathbf{S} + (\mathbf{n} \cdot \mathbf{c})t + d = 0 \Rightarrow t = \frac{-(\mathbf{n} \cdot \mathbf{S} + d)}{(\mathbf{n} \cdot \mathbf{c})}$$

28

Ray triangle intersection

- Tomas Möller and Ben Trumbore, "Fast, minimum storage ray-triangle intersection", *Journal of graphics tools*, 2(1): 21-28, 1997
- Barycentric coordinates

$$\mathbf{p} = (1-u-v)\mathbf{P}_0 + u\mathbf{P}_1 + v\mathbf{P}_2 \qquad u, v \geq 0, u+v \leq 1$$

  - **p** is a point in a triangle with vertices $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$
- If *r(t)* belongs to both the line and triangle

$$r(t) = \mathbf{S} + \mathbf{c}t = (1-u-v)\mathbf{P}_0 + u\mathbf{P}_1 + v\mathbf{P}_2$$

- Solve for *(t, u, v)*
  - If *(u, v)* complies with the restriction, then ray *r(t)* intersects the triangle
  - Using *t* the intersection point can be easily computed

Courtesy of http://www.lighthouse3d.com/opengl/maths/index.php?raytriint

Ray triangle intersection

- R. J. Segura, F. R. Feito,"Algorithms to test Ray-triangle Intersection Comparative Study", WSCG 2001

```
#define innerProduct(v,q) \
        ((v)[0] * (q)[0] + \
         (v)[1] * (q)[1] + \
         (v)[2] * (q)[2])
```

```
#define crossProduct(a,b,c) \
    (a)[0] = (b)[1] * (c)[2] - (c)[1] * (b)[2]; \
    (a)[1] = (b)[2] * (c)[0] - (c)[2] * (b)[0]; \
    (a)[2] = (b)[0] * (c)[1] - (c)[0] * (b)[1];
```

```
/* a = b - c */
#define vector(a,b,c) \
        (a)[0] = (b)[0] - (c)[0];       \
        (a)[1] = (b)[1] - (c)[1];       \
        (a)[2] = (b)[2] - (c)[2];


int rayIntersectsTriangle(float *p, float *d,
                          float *v0, float *v1, float *v2) {

        float e1[3],e2[3],h[3],s[3],q[3];
        float a,f,u,v;

        vector(e1,v1,v0);
        vector(e2,v2,v0);
        crossProduct(h,d,e2);
        a = innerProduct(e1,h);

        if (a > -0.00001 && a < 0.00001)
                return(false);

        f = 1/a;
        vector(s,p,v0);
        u = f * (innerProduct(s,h));

        if (u < 0.0 || u > 1.0)
                return(false);

        crossProduct(q,s,e1);
        v = f * innerProduct(d,q);
        if (v < 0.0 || u + v > 1.0)
                return(false);

        return(true);
}
```

Courtesy of http://www.lighthouse3d.com/opengl/maths/index.php?raytriint

Ray tracing flow

```
Color3 Scene :: shade(Ray& r)
{
  Get the first hit, and build hitInfo h
  Shape* myObj = (Shape*)h.hitObject; //pointer to the hit object
  Color3 color.set(the emissive component);
  color.add(ambient contribution);
  get the normalized normal vector m at the hit point
  for(each light source)
      add the diffuse and specular components
  // now add the reflected and transmitted components

  if(r.recurseLevel == maxRecursionLevel)
     return color; // don't recurse further

  if(hit object is shiny enough) // add any reflected light
  {
     get reflection direction
     build reflected ray, refl
     refl.recurseLevel = r.recurseLevel + 1;
     color.add(shininess * shade(refl));
  }
  if(hit object is transparent enough)
  {
     get transmited direction
     build transmitted ray, trans
     trans.recurseLevel = r.recurseLevel + 1;
     color.add(transparency * shade(trans));
  }
  return color;
}
```

Local Phong illumination

Figure out reflected/ refracted ray direction and recurse

Adapted from F.S. Hill and CISC 640/440, Fall 2005

References

- Textbooks
  - F. S. Hill, "Computer Graphics Using OpenGL"
- Commonly used ray tracing program (completely free and available for most platforms)
  - http://www.povray.org/
- Interesting Links
  - Interactive Ray Tracer – Alyosha Efros
- Ray Tracing explained
  - http://www.geocities.com/jamisbuck/raytracing.html
  - http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm

Multimedia Systems

- Copyright protection and authentication

Dr. Yi-Zhe Song

EBU706U

Today's agenda

- Digital Rights Management
- What is digital watermarking?
- Watermarking: properties

EBU706U

Digital Rights Management

- Digital Rights Management (DRM)
  - strategy used by publishers or copyright owners
    - to control access to and usage of digital data or hardware,
    - to implement restrictions associated with a specific instance of a digital work or device
- Copy protection
  - technologies that control or restrict the use and access of digital content on electronic devices with such technologies installed, acting as components of a DRM design

EBU706U

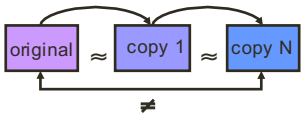
Properties of digital multimedia

- Digital media files
  - may be stored, copied, and distributed easily, rapidly, and with no loss of fidelity
  - Can be manipulated and edited easily and inexpensively
- Advent of PC + Internet + file sharing tools
  - Have made unauthorized sharing of digital files possible and profitable
  - Digital piracy

EBU706U

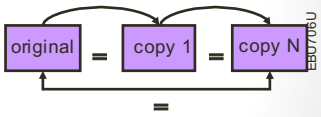
Analogue and digital multimedia

Analogue media
- photocopies
- audio cassettes
- photographs
- VHS videotapes



Built-in protection against copying and redistribution

Digital media
- ASCII, postscript, PDF
- MP3 audio
- JPEG images
- DVDs, MPEG video



No inherent protection against copying and redistribution

Free distribution net: Internet

EBU706U

DRM

- Digital rights management
  - refer to the protective schemes
  - limited to digital media
    - because of their special characteristics (exact copy)
  - most commonly used by the entertainment industry
    - films
    - recording
  - use in other media as well
    - online music stores (Apple's iTunes Store)
    - certain e-books producers
    - TV producers have begun demanding implementation of DRM measures to control access to the content of their shows, e.g. in connection with the TiVo system (digital video recorder)

EBU706U

MPEG REL

- MPEG REL (MPEG 21)
  - REL = Rights Expression Language
  - machine-readable language that declares rights and permissions
  - XML-based language for expressing rights related to the use and distribution of digital content as well as access to services
  - MPEG 21 specifies the syntax and semantics of a REL
  - defines an "authorisation model to specify whether the semantics of a set of Rights Expressions permit a given Principal to perform a given Right upon a given optional Resource during a given time interval based on a given authorization context and a given trust root"

EBU706U

MPEG REL

- MPEG REL
  - Simple and extensible data model for many of its key concepts and elements
  - includes four basic entities whose relationship between entities is defined by the MPEG REL assertion grant
  - a grant consists of the following elements
    - The principal to whom the grant is issued
    - The right that the grant specifies
    - The resource to which the right in the grant applies
    - The condition that must be met before the right can be exercised
  - A full rights expression is called a license
    - one or more grants and an issuer, which identifies the party who issued the license

EBU706U

Traditional methods for data protection

Access-control headers → easily removed/altered

Encryption → decrypted data unprotected

Copy protection → susceptible to hacking

EBU706U

Today's agenda

- Digital Rights Management
- What is digital watermarking?
- Watermarking: properties

EBU706U

Digital watermarking

- Definition
  - The imperceptible, robust and secure communication of information by embedding it in an original digital document and retrieving it from the processed version of that document
- Note (what is an "attack" on watermarked data?)
  - Watermarked data is likely to be processed (e.g. compressed)
  - Attack: any processing that may impair watermark

EBU706U

What is the watermark?

Other definitions:

- The watermark is a signal
  - The watermarked image/video/song is the channel or carrier
  - Attacks are noise
- The watermark is the embedded information
  - It can be retrieved or
  - It can be detected or
  - It triggers a given action to protect, disable or uncover tampering

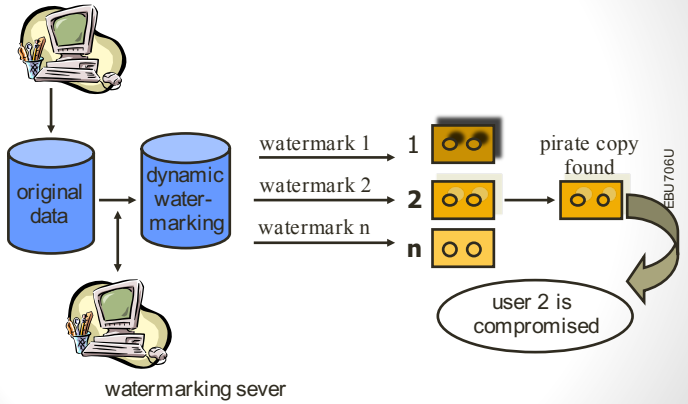
EBU706U

Watermark classification

- Robust watermarking
  - Embedding signatures for ownership protection
  - To carry metadata
  - To carry any additional information (e.g. steganography)
- Semi-Fragile watermarking
  - Authentication under adverse conditions (i.e. can resist to some processing on the data)
- Fragile watermarking
  - Conventional authentication (the watermark will break if any processing is attempted on the data)

EBU706U

Example: distribution from a library



EBU706U

Today's agenda

- Digital Rights Management
- What is digital watermarking?
- Watermarking: properties

EBU706U

General properties

- Imperceptibility
  - Watermarked data and original data should be perceptually indistinguishable
- Robustness (required for ownership protection)
  - embedded information should remain in the image/video/song even after any processing, transformation or attack
  - processing of the watermarked data cannot damage or destroy the embedded information without making the processed data useless
  - embedded information cannot be detected, read, and/or modified by unauthorized parties
  - Kerckhoff's principle should be satisfied

EBU706U

Kerckhoff's principle

- Kerckhoff's principle
  - "Security resides in the secrecy of the key, not in the secrecy of the algorithm"
  - Taken from cryptography: it is assumed that your opponent has complete knowledge of your strategy but lacks a secret key

Strategy = algorithm & implementation

Watermarking = strategy + secret key

EBU706U

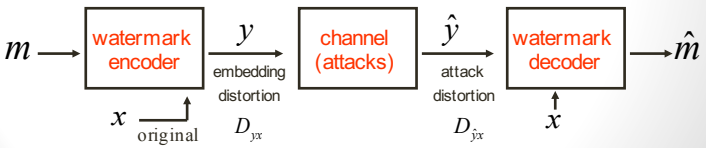
Robust watermarking: goals

- To watermark the digital content (e.g., image, audio) without degrading its quality
  - by modifying LSB or low-order bits; blue channel
- To achieve robustness with respect to
  - Compression techniques
    - JPEG, JPEG2000, MPEG-2, MP3
  - Format conversions
    - changing the video frame rate
  - Geometric transformations
    - scaling, cropping an image
  - Fraudulent attacks
  - Reproduction
    - printing, photocopying, and scanning

EBU706U

Evaluating robustness

- Robustness
  - A watermark is robust if communication cannot be impaired without rendering the attacked data useless
  - Communication is impaired if the watermark is no longer reliably detectable/decodable
  - Attacked data becomes useless if after the attack strong distortions are perceived

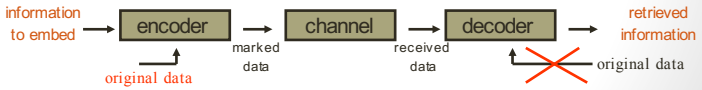


EBU706U

Blind watermarking

- Reception-with-original watermarking
  - original data assists in watermarking decoding

- Blind watermarking → receiver attempts to decode the watermark information without knowledge of original data



EBU706U

What did we learn today?

- Digital Rights Management
- What is digital watermarking?
- Watermarking: properties

EBU706U

Questions

In digital watermarking technology:

i) What are the two basic properties when copyright protection is targeted?

ii) State Kerckhoff's principle and explain its use with watermarking.

iii) What is the main difference between watermarking for copyright protection and watermarking for authentication?

EBU706U

iii) What is the main difference between watermarking for copyright protection and watermarking for authentication?

- Copyright protection requires robust watermarking, so the watermark can be retrieved (unmodified) from the data even after processing and transformation.
- Authentication requires fragile or semi-fragile watermarks. Fragile watermarks can serve as proof of authenticity, since they will be damaged or fully destroyed by any transformation (fragile watermark) or specific transformations (semi-fragile watermarks) of the watermarked document.

EBU706U