

INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 8: “Engine Fundamentals”

Welcome!



Rendering

Topics covered so far:

Basics:

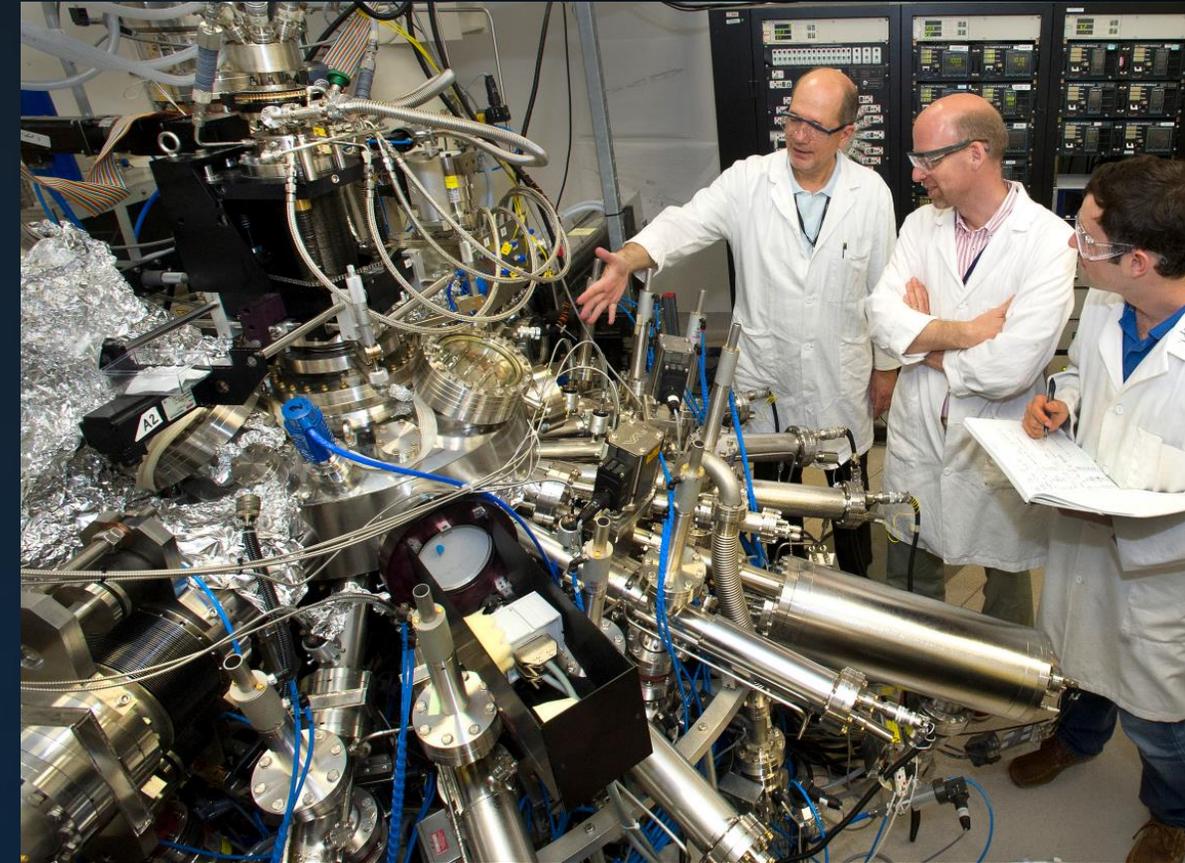
- Rasters
- Vectors
- Color representation

Ray tracing:

- Light transport
- Camera setup
- Textures

Shading:

- $N \cdot L$
- Distance attenuation
- Pure specular



Rendering

Rendering – Functional overview

1. Transform:
translating / rotating / scaling meshes
2. Project:
calculating 2D screen positions
3. Rasterize:
determining affected pixels
4. Shade:
calculate color per affected pixel

```

...
    & (depth < MAXDEPTH)
...
    c = inside / 1.0;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (cos2t
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse;
estimation - doing it properly, check
if;
radiance = SampleLight( @rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (survive
...
y = true;
at b
at3
at w
at c
at c
E *
...
andc
yive
...
at3
surv
pdf
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```

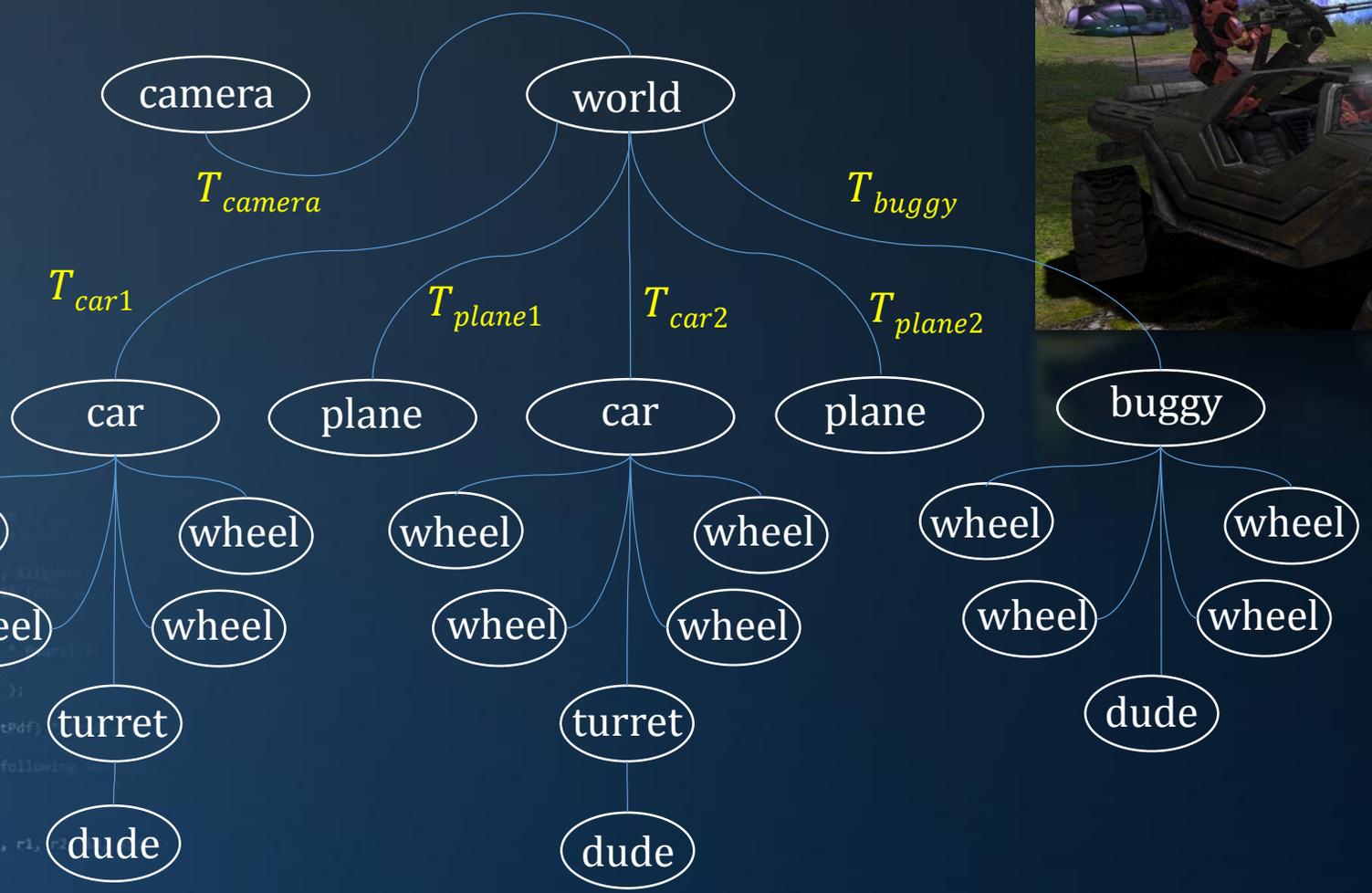




Rendering

Rendering – Data Overview

```
...  
& (depth < MAXDEPTH)  
...  
c = inside / ...  
nt = nt / nc; ...  
pos2t = 1.0f - nnt; ...  
D, N );  
...  
at a = nt - nc, b = nt ...  
at Tr = 1 - (R0 + (1 - R0) ...  
Tr) R = (D * nnt - N * ...  
...  
E * diffuse;  
= true;  
...  
efl + refr) && (depth < MAXDEPTH)  
...  
D, N );  
-efl * E * diffuse;  
= true;  
...  
MAXDEPTH)  
survive = SurvivalProbability( ...  
estimation - doing ...  
if;  
-radiance = SampleLight( &rand, ...  
e.x + radiance.y + radiance.z);  
...  
v = true;  
at brdfPdf = EvaluateDiffuse( L, N );  
at3 factor = diffuse * INVPI;  
at weight = Mis2( directPdf, brdfPdf );  
at cosThetaOut = dot( N, L );  
E * ((weight * cosThetaOut) / directPdf);  
...  
random walk - done properly, closely following ...  
ive)  
...  
at3 brdf = SampleDiffuse( diffuse, N, r1, ...  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
ision = true;
```



Rendering

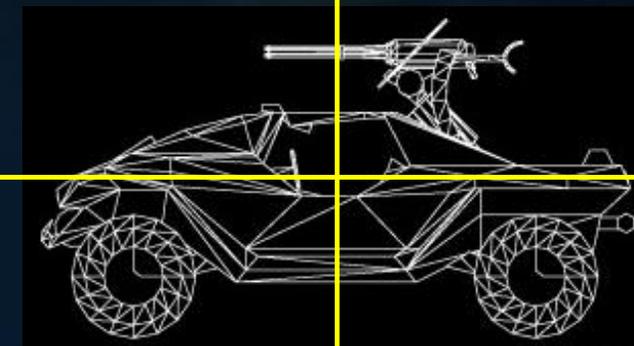
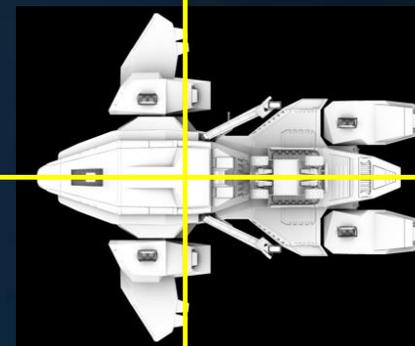
Rendering – Data Overview

Objects are organized in a hierarchy: the *scenegraph*.

In this hierarchy, objects have translations and orientations relative to their parent node.

Relative translations and orientations are specified using matrices.

Mesh vertices are defined in a coordinate system known as *object space*.



Rendering

Rendering – Data Overview

Transform takes our meshes from object space (3D) to camera space (3D).

Project takes the vertex data from camera space (3D) to screen space (2D).



```

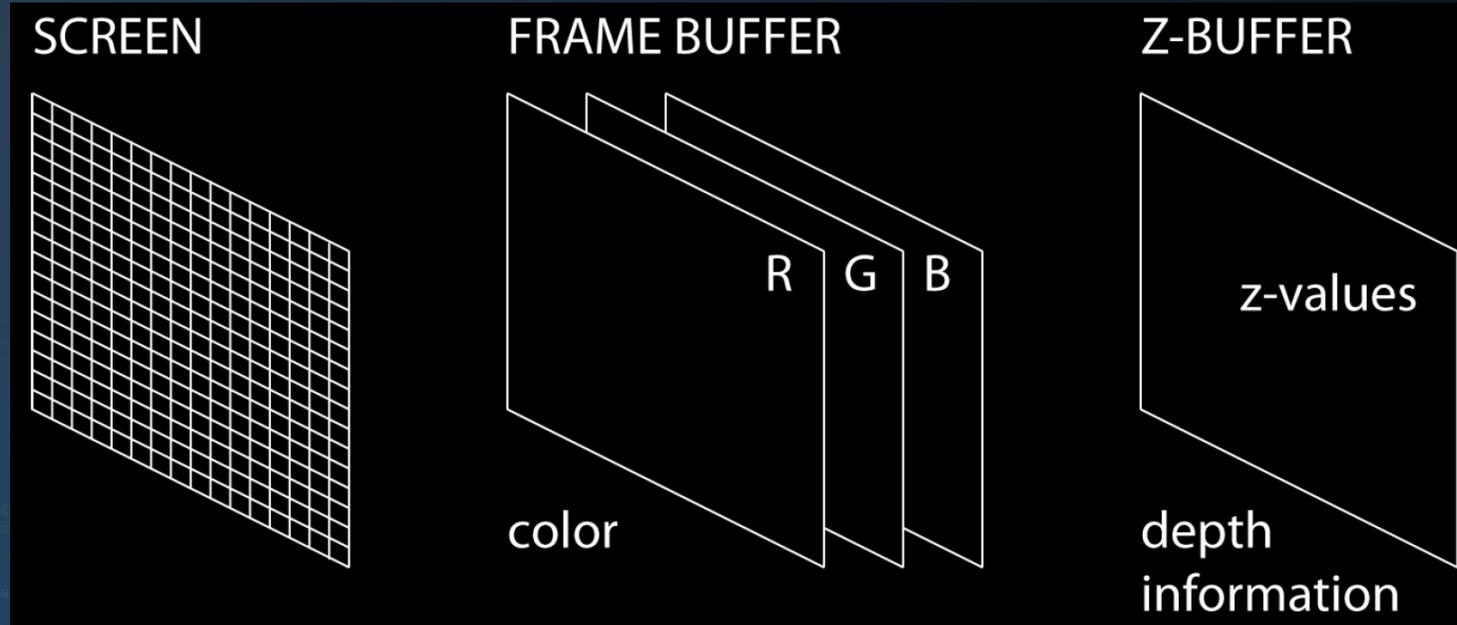
void RayTracer::Render(const Ray & ray, unsigned short *depth, unsigned short *Xi, unsigned short *Zi, unsigned short *Xi2, unsigned short *Zi2, unsigned short *Xi3, unsigned short *Zi3, unsigned short *Xi4, unsigned short *Zi4, unsigned short *Xi5, unsigned short *Zi5, unsigned short *Xi6, unsigned short *Zi6, unsigned short *Xi7, unsigned short *Zi7, unsigned short *Xi8, unsigned short *Zi8, unsigned short *Xi9, unsigned short *Zi9, unsigned short *Xi10, unsigned short *Zi10, unsigned short *Xi11, unsigned short *Zi11, unsigned short *Xi12, unsigned short *Zi12, unsigned short *Xi13, unsigned short *Zi13, unsigned short *Xi14, unsigned short *Zi14, unsigned short *Xi15, unsigned short *Zi15, unsigned short *Xi16, unsigned short *Zi16, unsigned short *Xi17, unsigned short *Zi17, unsigned short *Xi18, unsigned short *Zi18, unsigned short *Xi19, unsigned short *Zi19, unsigned short *Xi20, unsigned short *Zi20, unsigned short *Xi21, unsigned short *Zi21, unsigned short *Xi22, unsigned short *Zi22, unsigned short *Xi23, unsigned short *Zi23, unsigned short *Xi24, unsigned short *Zi24, unsigned short *Xi25, unsigned short *Zi25, unsigned short *Xi26, unsigned short *Zi26, unsigned short *Xi27, unsigned short *Zi27, unsigned short *Xi28, unsigned short *Zi28, unsigned short *Xi29, unsigned short *Zi29, unsigned short *Xi30, unsigned short *Zi30, unsigned short *Xi31, unsigned short *Zi31, unsigned short *Xi32, unsigned short *Zi32, unsigned short *Xi33, unsigned short *Zi33, unsigned short *Xi34, unsigned short *Zi34, unsigned short *Xi35, unsigned short *Zi35, unsigned short *Xi36, unsigned short *Zi36, unsigned short *Xi37, unsigned short *Zi37, unsigned short *Xi38, unsigned short *Zi38, unsigned short *Xi39, unsigned short *Zi39, unsigned short *Xi40, unsigned short *Zi40, unsigned short *Xi41, unsigned short *Zi41, unsigned short *Xi42, unsigned short *Zi42, unsigned short *Xi43, unsigned short *Zi43, unsigned short *Xi44, unsigned short *Zi44, unsigned short *Xi45, unsigned short *Zi45, unsigned short *Xi46, unsigned short *Zi46, unsigned short *Xi47, unsigned short *Zi47, unsigned short *Xi48, unsigned short *Zi48, unsigned short *Xi49, unsigned short *Zi49, unsigned short *Xi50, unsigned short *Zi50, unsigned short *Xi51, unsigned short *Zi51, unsigned short *Xi52, unsigned short *Zi52, unsigned short *Xi53, unsigned short *Zi53, unsigned short *Xi54, unsigned short *Zi54, unsigned short *Xi55, unsigned short *Zi55, unsigned short *Xi56, unsigned short *Zi56, unsigned short *Xi57, unsigned short *Zi57, unsigned short *Xi58, unsigned short *Zi58, unsigned short *Xi59, unsigned short *Zi59, unsigned short *Xi60, unsigned short *Zi60, unsigned short *Xi61, unsigned short *Zi61, unsigned short *Xi62, unsigned short *Zi62, unsigned short *Xi63, unsigned short *Zi63, unsigned short *Xi64, unsigned short *Zi64, unsigned short *Xi65, unsigned short *Zi65, unsigned short *Xi66, unsigned short *Zi66, unsigned short *Xi67, unsigned short *Zi67, unsigned short *Xi68, unsigned short *Zi68, unsigned short *Xi69, unsigned short *Zi69, unsigned short *Xi70, unsigned short *Zi70, unsigned short *Xi71, unsigned short *Zi71, unsigned short *Xi72, unsigned short *Zi72, unsigned short *Xi73, unsigned short *Zi73, unsigned short *Xi74, unsigned short *Zi74, unsigned short *Xi75, unsigned short *Zi75, unsigned short *Xi76, unsigned short *Zi76, unsigned short *Xi77, unsigned short *Zi77, unsigned short *Xi78, unsigned short *Zi78, unsigned short *Xi79, unsigned short *Zi79, unsigned short *Xi80, unsigned short *Zi80, unsigned short *Xi81, unsigned short *Zi81, unsigned short *Xi82, unsigned short *Zi82, unsigned short *Xi83, unsigned short *Zi83, unsigned short *Xi84, unsigned short *Zi84, unsigned short *Xi85, unsigned short *Zi85, unsigned short *Xi86, unsigned short *Zi86, unsigned short *Xi87, unsigned short *Zi87, unsigned short *Xi88, unsigned short *Zi88, unsigned short *Xi89, unsigned short *Zi89, unsigned short *Xi90, unsigned short *Zi90, unsigned short *Xi91, unsigned short *Zi91, unsigned short *Xi92, unsigned short *Zi92, unsigned short *Xi93, unsigned short *Zi93, unsigned short *Xi94, unsigned short *Zi94, unsigned short *Xi95, unsigned short *Zi95, unsigned short *Xi96, unsigned short *Zi96, unsigned short *Xi97, unsigned short *Zi97, unsigned short *Xi98, unsigned short *Zi98, unsigned short *Xi99, unsigned short *Zi99)
    
```



Rendering

Rendering – Data Overview

The screen is represented by (at least) two buffers:



Rendering

Rendering – Components

Scenegraph

Culling

Lecture 11

Vertex transform pipeline

Matrices to convert from one space to another

Lecture 8

Perspective

Lecture 9

Rasterization

Interpolation

Clipping

Depth sorting: z-buffer

Lecture 11

Lecture 11

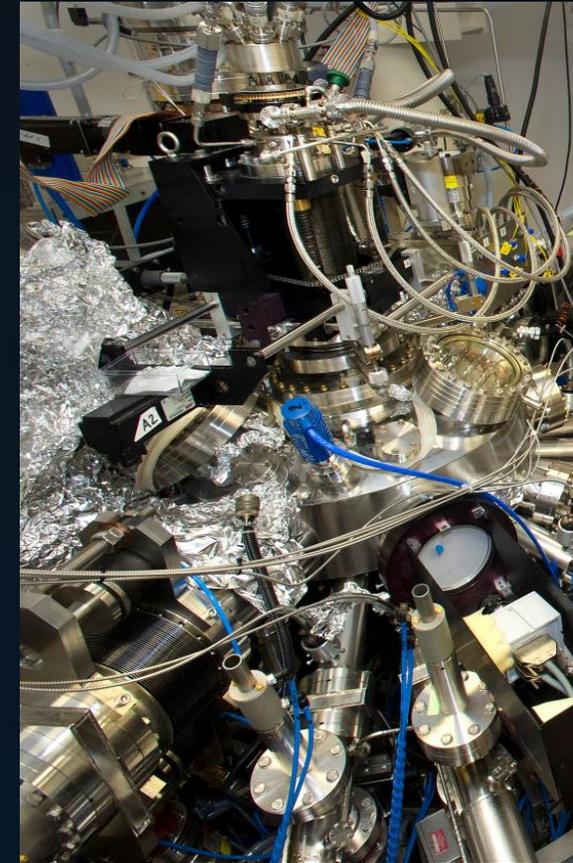
Shading

Light / material interaction

Complex materials

P2

P3



Matrices

Bases in \mathbb{R}^2 and \mathbb{R}^3

Recall:

- Two linearly independent vectors form a base.
- We can reach any point in using:

$$\vec{a} = \lambda_1 \vec{u} + \lambda_2 \vec{v}$$

- If \vec{u} and \vec{v} are perpendicular unit vectors, the base is *orthonormal*.
- The Cartesian coordinate system is an example of this, with $\vec{u} = (1,0)$ and $\vec{v} = (0,1)$.

By manipulating \vec{u} and \vec{v} , we can create a ‘coordinate system’ within a coordinate system.



Matrices

Bases in \mathbb{R}^2 and \mathbb{R}^3

This extends naturally to \mathbb{R}^3 :

Three vectors, \vec{u} , \vec{v} and \vec{w} allow us to reach any point in 3D space;

$$a = \lambda_1 \vec{u} + \lambda_2 \vec{v} + \lambda_3 \vec{w}$$

Again, manipulating \vec{u} , \vec{v} and \vec{w} changes where coordinates specified as $(\lambda_1, \lambda_2, \lambda_3)$ end up.



```

ics
& (depth < MAXDEPTH)
{
  t = inside / (inside + outside);
  nt = nt / nc;
  cos2t = 1.0f - nnt;
  D, N );
}
at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * t);
Tr) R = (D * nnt - N * (cos2t
);
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
efl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability( diffuse *
estimation - doing it properly, closely
if;
radiance = SampleLight( @rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (depth <
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pdf;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following death
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Matrices

Matrices

A vector is an ordered set of d scalar values (i.e., a d -tuple):

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \text{ or } (v_1, v_2, v_3) \text{ or ...}$$

A $m \times n$ matrix is an array of $m \cdot n$ scalar values, sorted in m rows and n columns:

$$M = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

The elements a_{ij} are referred to as the *coefficients* of the matrix (or *elements*, *entries*). Note that here i is the row; j is the column.

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (inside + outside);
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * t);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    -refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, check
if;
radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (cos
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pearls
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Matrices

Terminology – Special Matrices

- A *diagonal matrix* is a matrix for which all elements a_{ij} are zero if $i \neq j$.
- An *identity matrix* is a diagonal matrix where each element $a_{ii} = 1$.
- The *zero matrix* contains only zeroes.

$$A = \begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 0.99 & 0 \\ 0 & 0 & 3.14 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

x y z

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Before we continue, what *is* a matrix?

- Just a group of numbers;
- In graphics: often a representation of a coordinate system.



Matrices

Matrices - Operations

Matrix addition is defined as:

$$A = B + C, \text{ with: } c_{ij} = a_{ij} + b_{ij}$$

Note that addition is only defined for matrices with the same dimensions.

Example:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 \\ 4 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 4 & 5 \end{pmatrix}$$

Subtraction works the same.

```

...
    & (depth < MAXDEPTH)
...
    c = inside / nc;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc; b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
radiance = SampleLight( @rand, I, M, Align
e.x + radiance.y + radiance.z) > 0) && (survive)
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Pearline;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Matrices

Matrices - Operations

Multiplying a matrix with a scalar is defined as follows:

$$A = \lambda B, \text{ with: } a_{ij} = \lambda b_{ij}$$

Example:

$$2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (inside + outside);
    nt = nt / nc; nde = nde / n;
    cos2t = 1.0f - nnt * nnd;
    D, N );
    )
...
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * t);
    Tr) R = (D * nnt - N * (nde
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    -refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
    radiance = SampleLight( &rand, I, M, Alignment,
    e.x + radiance.y + radiance.z) > 0) && (cosThetaOut
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * survive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Matrices

Matrices - Operations

Multiplying a matrix (dimensions $w_A \times h_A$) with another matrix (dimensions $w_B \times h_B$):

$$C = AB, \text{ with: } c_{ij} = \sum_{k=1}^{w_A} a_{ik} b_{kj}$$

Example:

$$\begin{pmatrix} 2 & 6 & 1 \\ 5 & 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} = \begin{pmatrix} 17 & 44 \\ 21 & 54 \end{pmatrix}$$

Note the dimensions of the resulting matrix: $h_A \times w_B$.

Matrix multiplication is only defined if $h_A = w_B$ (i.e., the width of B is equal to the height of A).

$$c_{11} = \sum_{k=1}^3 a_{1k} b_{k1} = 2 * 1 + 6 * 2 + 1 * 3 = 17$$

$$c_{21} = \sum_{k=1}^3 a_{2k} b_{k1} = 5 * 1 + 2 * 2 + 4 * 3 = 21$$

$$c_{12} = \sum_{k=1}^3 a_{1k} b_{k2} = 2 * 4 + 6 * 5 + 1 * 6 = 44$$

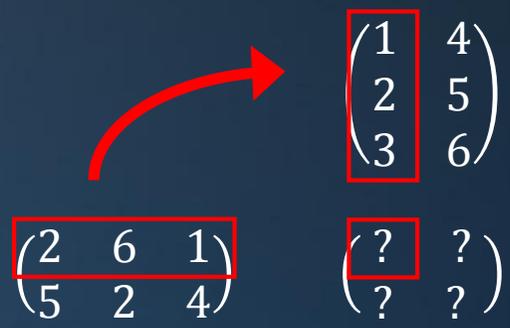
$$c_{22} = \sum_{k=1}^3 a_{2k} b_{k2} = 5 * 4 + 2 * 5 + 4 * 6 = 54$$



Matrices

Matrices - Operations

Doing matrix multiplication manually:



Note that each cell in the resulting matrix is essentially the dot product of a row and a column.

Some properties:

Matrix multiplication is distributive over addition:

$$A(B + C) = AB + AC$$

$$(A + B)C = AC + BC$$

...and associative:

$$(AB)C = A(BC)$$

However, matrix multiplication is not commutative, i.e., in general:

$$AB \neq BA$$



Matrices

Matrices - Operations

Doing matrix multiplication manually:

$$\begin{pmatrix} 2 & 6 & 1 \\ 5 & 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

$$\begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix}$$

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

Multiplying by the zero matrix yields the zero matrix:

$$0A = A0 = 0$$

Multiplying by the identity matrix yields the original matrix:

$$IA = AI = A$$



Matrices

Matrices - Operations

The *transpose* A^T of an $m \times n$ matrix is an $n \times m$ matrix that is obtained by interchanging rows and columns: a_{ij} becomes a_{ji} for all i, j :

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad A^T = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}$$

The transpose of the product of two matrices is:

$$(AB)^T = B^T A^T$$

```

...
    & (depth < MAXDEPTH)
...
    inside / ray;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse
estimation - doing it properly, close
if;
    radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (survive)
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Pea
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Matrices

Matrices - Operations

We can multiply a d -dimensional vector by an $m \times d$ matrix:

$$\begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{md} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = \begin{pmatrix} a_{11}v_1 + \cdots + a_{1d}v_d \\ \cdots + \cdots + \cdots \\ a_{m1}v_1 + \cdots + a_{md}v_d \end{pmatrix}$$

Note:

This is the same as matrix concatenation; the vector is simply an $m \times 1$ matrix.

Example: multiply a 3D vector by a 3x3 matrix:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y + a_{13}z \\ a_{21}x + a_{22}y + a_{23}z \\ a_{31}x + a_{32}y + a_{33}z \end{pmatrix}$$



Matrices

Matrices - Operations

We can multiply a d -dimensional vector by an $m \times d$ matrix:

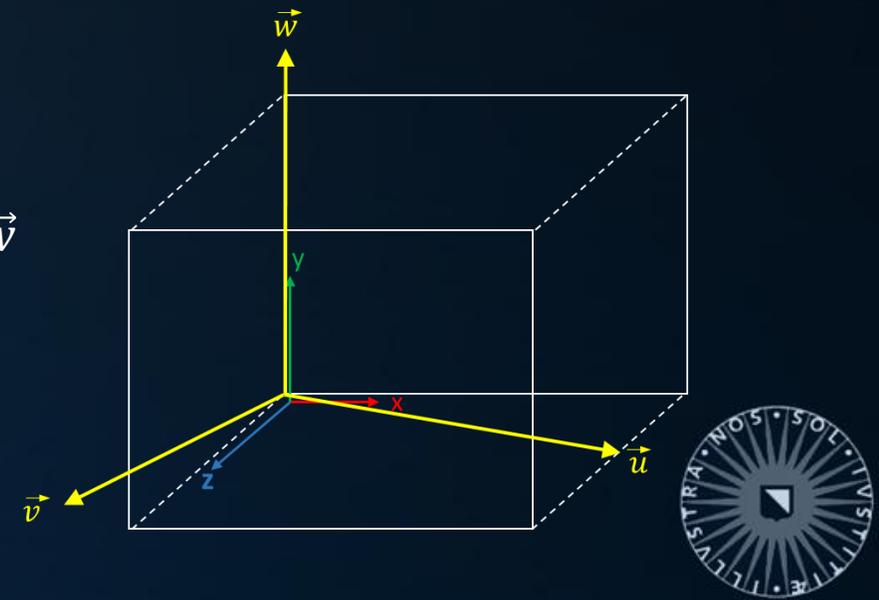
$$\begin{pmatrix} a_{11} & \dots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{md} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = \begin{pmatrix} a_{11}v_1 + \dots + a_{1d}v_d \\ \dots + \dots + \dots \\ a_{m1}v_1 + \dots + a_{md}v_d \end{pmatrix}$$

Note:

This is the same as matrix concatenation; the vector is simply an $m \times 1$ matrix.

Example: multiply a 3D vector by a 3x3 matrix:

$$\begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} u_x x + v_x y + w_x z \\ u_y x + v_y y + w_y z \\ u_z x + v_z y + w_z z \end{pmatrix} = x\vec{u} + y\vec{v} + z\vec{w}$$



Matrices

Matrices – Determinant

The determinant $|A|$ of an $n \times n$ matrix A is the signed area or volume spanned by its column vectors.

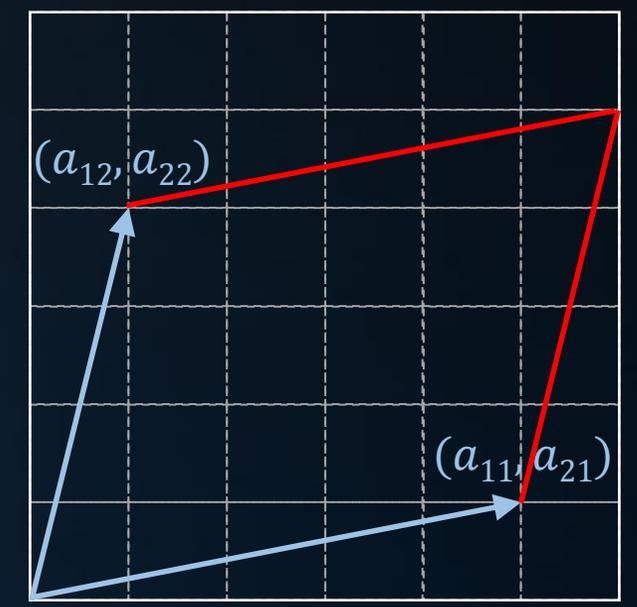
Example (in \mathbb{R}^2):

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \det A = |A| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

In this case, the determinant is the oriented area of the *parallelogram* defined by the two column vectors.

The determinant is positive if the vectors are counter-clockwise, or negative if they are clockwise. Therefore:

$$\det \begin{vmatrix} \vec{a}_1 & \vec{a}_2 \end{vmatrix} = -\det \begin{vmatrix} \vec{a}_2 & \vec{a}_1 \end{vmatrix}$$



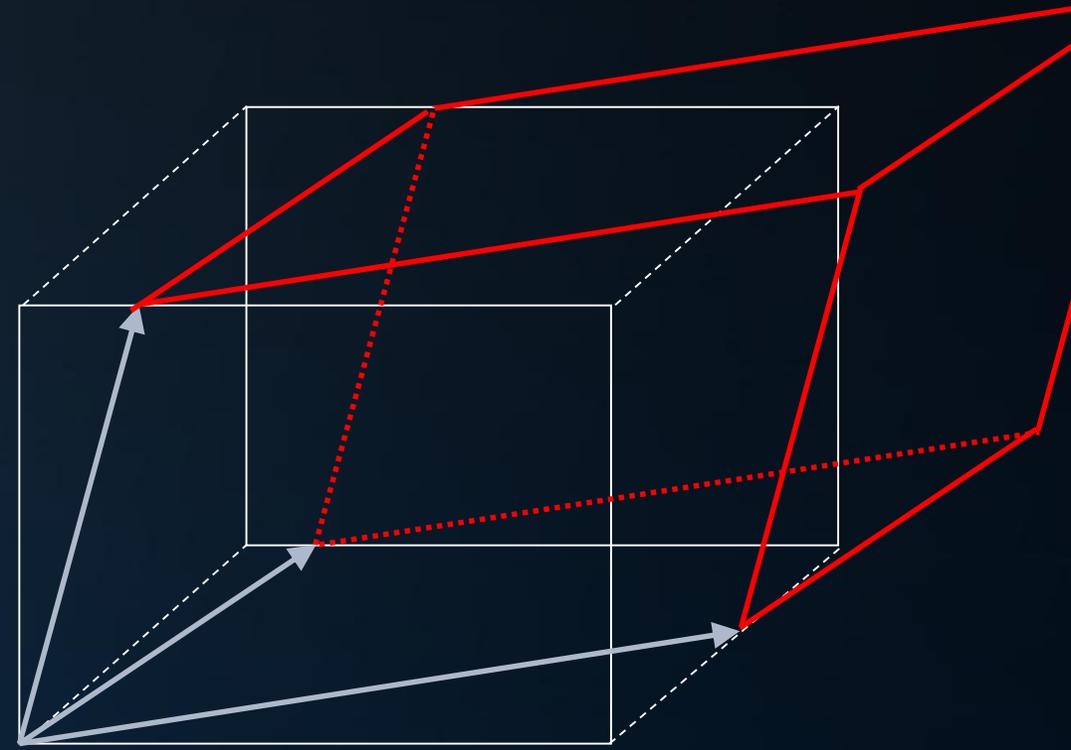
Matrices

Matrices – Determinant

The determinant $|A|$ of an $n \times n$ matrix A is the signed volume spanned by its column vectors.

In \mathbb{R}^3 , the determinant is the oriented area of the parallelepiped defined by the three column vectors.

$$\det A = |A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$



```

...
    & (depth < MAXDEPTH)
...
    inside / outside
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0)
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) && (rand <
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Matrices

Matrices – Determinant

Calculating determinants: Laplace’s expansion.

The determinant of a matrix is the sum of the products of the elements of any row or column of the matrix with their *cofactors*.

The cofactor of an entry a_{ij} in an $n \times n$ matrix A is:

- The determinant of the $(n - 1) \times (n - 1)$ matrix A' ,
- that is obtained from A by removing the i -th row and j -th column,
- multiplied by -1^{i+j} .

Example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$a_{11}^c = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} * (-1^2)$$

$$a_{12}^c = \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} * (-1^3)$$

$$a_{13}^c = \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} * (-1^4)$$

$$|A| = a_{11} a_{11}^c + a_{12} a_{12}^c + a_{13} a_{13}^c$$



Matrices

Matrices – Determinant

Full example for 3 × 3 matrix:

$$\begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix} = 0 \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} - 1 \begin{vmatrix} 3 & 5 \\ 6 & 8 \end{vmatrix} + 2 \begin{vmatrix} 3 & 4 \\ 6 & 7 \end{vmatrix}$$

$$\begin{vmatrix} 3 & 5 \\ 6 & 8 \end{vmatrix} = 3 * 8 * -1^2 + 5 * 6 * -1^3 = -6$$

$$\begin{vmatrix} 3 & 4 \\ 6 & 7 \end{vmatrix} = 3 * 7 * -1^2 + 4 * 6 * -1^3 = -3$$

$$0 - 1 * -6 + 2 * -3 = 0.$$

Generic approach for a for 3 × 3 matrix:

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - \dots$$

$$= (aei + bfg + cdh) - (ceg + afh + bdi)$$



Rule of Sarrus for 2 × 2: $\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$



Matrices

Matrices – Adjoint

The *adjoint* (or *adjugate*) \tilde{A} of matrix A is the transpose of the cofactor matrix of A .

Example:

$$A = \begin{pmatrix} 2 & 5 \\ 1 & 3 \end{pmatrix} \rightarrow C = \begin{pmatrix} 3 * (-1^2) & 1 * (-1^3) \\ 5 * (-1^3) & 2 * (-1^4) \end{pmatrix} = \begin{pmatrix} 3 & -1 \\ -5 & 2 \end{pmatrix}$$

$$adj(A) = C^T = \begin{pmatrix} 3 & -5 \\ -1 & 2 \end{pmatrix}.$$

The cofactor of an entry a_{ij} in an $n \times n$ matrix A is:

- The determinant of the $(n - 1) \times (n - 1)$ matrix A' ,
- that is obtained from A by removing the i -th row and j -th column,
- multiplied by -1^{i+j} .

```

...
    & (depth < MAXDEPTH)
...
    inside / nc;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    -refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, check
if;
radiance = SampleLight( @rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (survive)
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, @R, @pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Matrices

Matrices – Inverse

The adjoint is used to calculate the inverse A^{-1} of a matrix A :

$$A^{-1} = \frac{\tilde{A}}{|A|}$$

```

...
    & (depth < MAXDEPTH)
...
    t = inside / (nc + inside);
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * t);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, M, &light;
e.x + radiance.y + radiance.z) && (rand.N
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

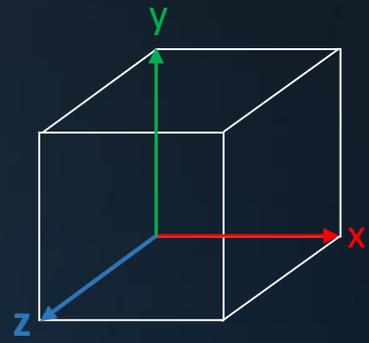
```



Matrices

Matrices – Overview

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



$n \times m$: n rows, m columns

$$\det(A) = |A| = 1 = (aei + bfg + cdh) - (ceg + afh + bdi)$$



$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$



note: $\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix} = -1$, and: $\det|\vec{a}_1 \vec{a}_2| = -\det|\vec{a}_2 \vec{a}_1|$

$$\text{cofactor } a_{11}^c = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} * (-1^2)$$

Adjoint \tilde{A} of A is C^T ; inverse A^{-1} is $\frac{\tilde{A}}{|A|}$.



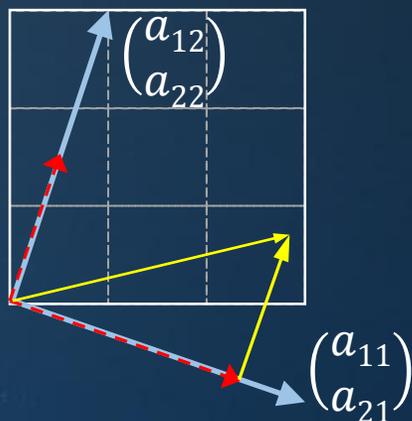
Transforms

Spaces - Introduction

As we have seen before, we can multiply a matrix with a vector.

In 2D:
$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{pmatrix} = x \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix} + y \begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}$$

In 3D:
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y + a_{13}z \\ a_{21}x + a_{22}y + a_{23}z \\ a_{31}x + a_{32}y + a_{33}z \end{pmatrix} = x \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} + y \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix} + z \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \end{pmatrix}$$



Geometric interpretation:

scalar multiplication of $\begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}$ by x , plus
 scalar multiplication of $\begin{pmatrix} a_{12} \\ a_{22} \end{pmatrix}$ by y yields
transformed point.

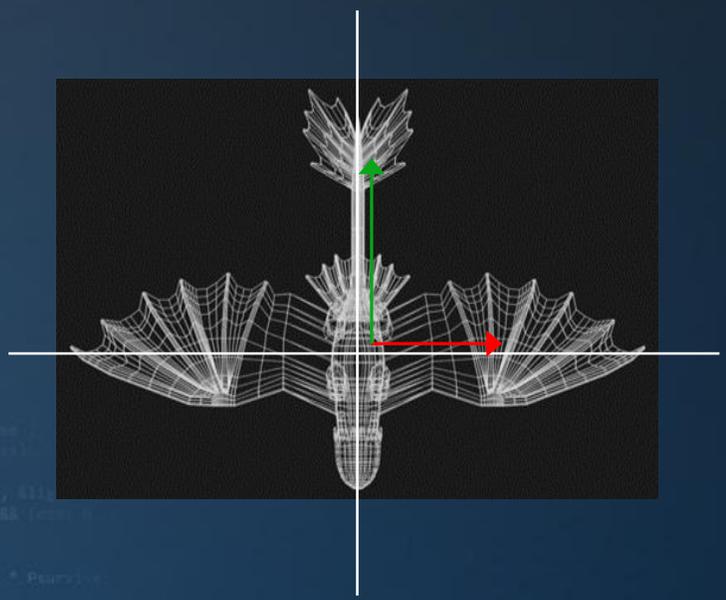


Transforms

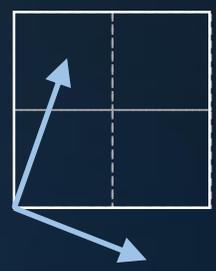
Spaces – Introduction

A matrix allows us to *transform* a coordinate system.

```
...ics
& (depth < MAXDEPTH)
...
t = inside / (inside + outside);
nt = nt / nc;
os2t = 1.0f - nnt;
D, N );
...
at a = nt - nc, b = nt;
at Tr = 1 - (R0 + (1 - R0) * t);
Tr) R = (D * nnt - N * (a * t + b * (1 - t)));
...
E * diffuse;
= true;
...
efl + refr) && (depth < MAXDEPTH)
D, N );
efl * E * diffuse;
= true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, M, All);
e.x + radiance.y + radiance.z) > 0) && (depth <
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Survival;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

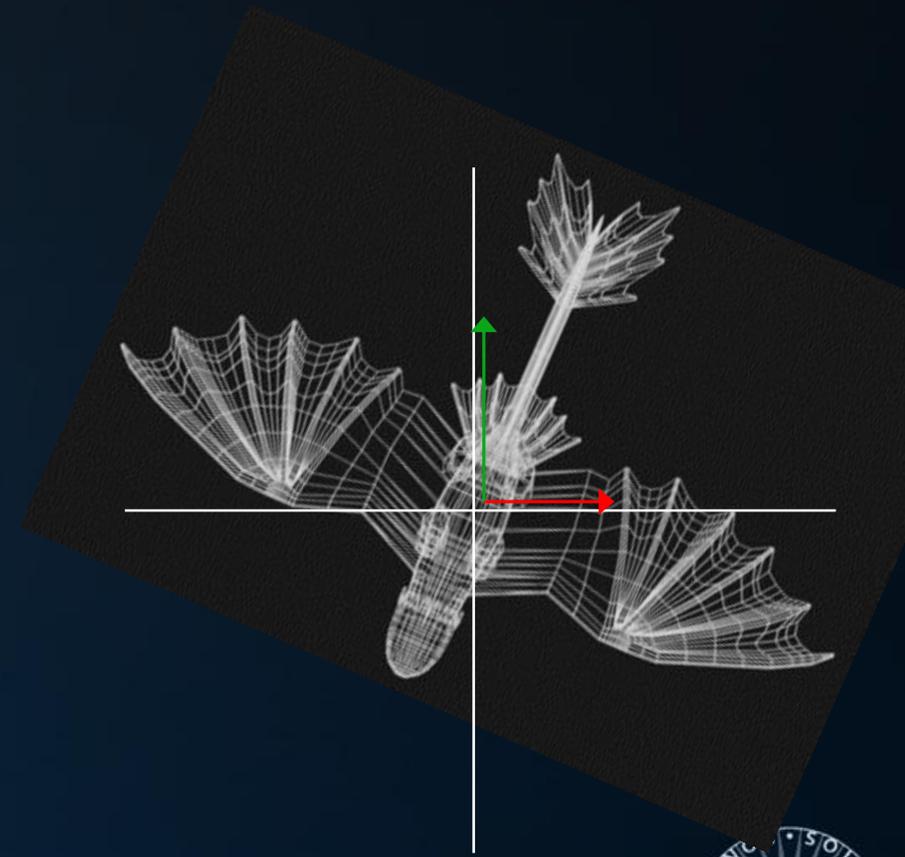


×



=

rotation
+
scale



Transforms

Spaces – Scaling

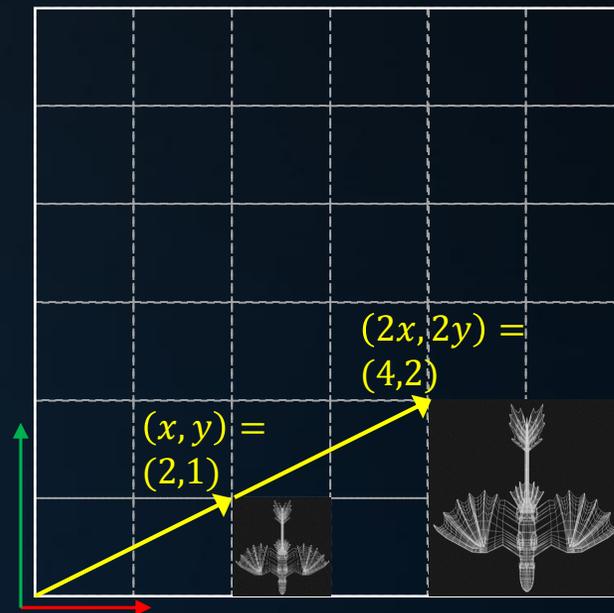
To scale by a factor 2 with respect to the origin, we apply the matrix

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Applied to a vector, we get:

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2x + 0y \\ 0x + 2y \end{pmatrix} = \begin{pmatrix} 2x \\ 2y \end{pmatrix}$$

This is called *uniform scaling*.



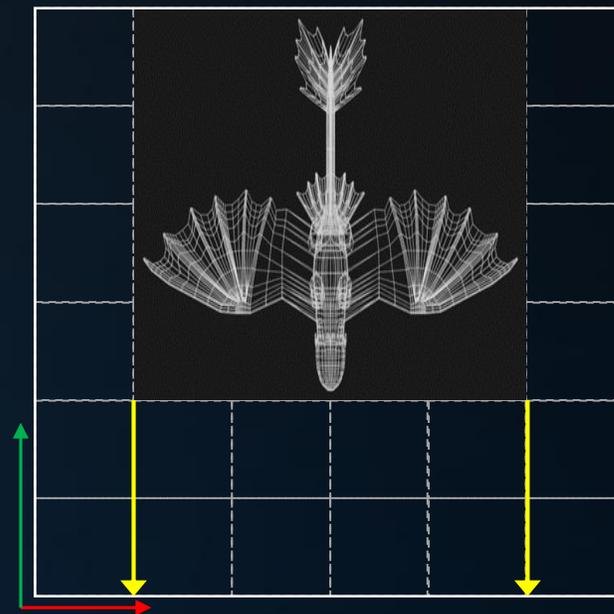
Transforms

Spaces – Projection

If we set one of the a_{ii} to 0, we get an *orthographic projection*.

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

This is useful for projecting a shadow of the dragon on the x-axis, or to draw a 3D object on a 2D screen.

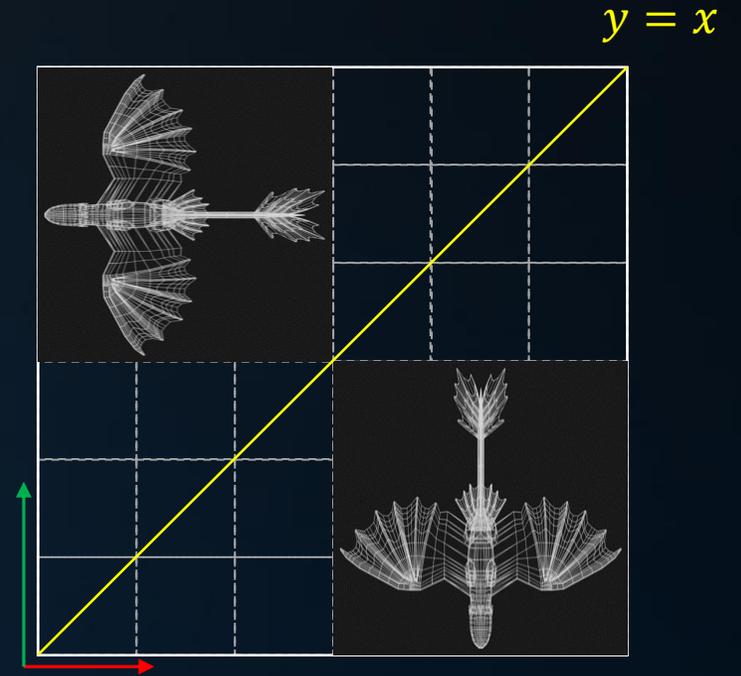


Transforms

Spaces – Reflection

We can construct a matrix that will swap x and y coordinates to get a reflection in the line $y = x$:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0x + 1y \\ 1x + 0y \end{pmatrix} = \begin{pmatrix} y \\ x \end{pmatrix}$$



```

...
    & (depth < MAXDEPTH)
...
    c = inside / 1.0;
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * c);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse *
estimation - doing it properly, closely
if;
    radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (survive)
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Survival;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Transforms

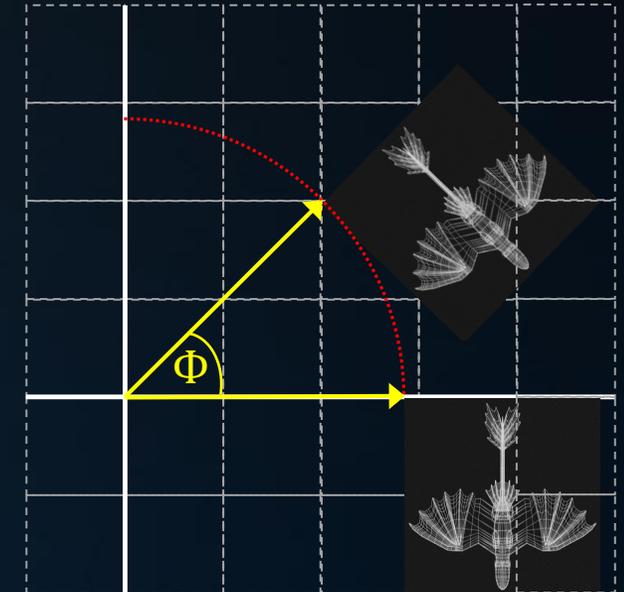
Spaces – Rotation

To rotate counter-clockwise about the origin, we use the following matrix:

$$\begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$$

For clockwise rotation, we use

$$\begin{pmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{pmatrix}$$



```

...
    & (depth < MAXDEPTH)
...
    inside / outside
    nt = nt / nc;
    cos2t = 1.0f - nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * nnt);
    Tr) R = (D * nnt - N * (a *
...
    E * diffuse;
    = true;
...
    refl + refr) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, M, Alignment,
e.x + radiance.y + radiance.z) > 0) && (rand <
...
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * survive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * radiance;
...
random walk - done properly, closely following death
ive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



Transforms

Spaces – Linear transformations

A function $T: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called a linear transformation, if it satisfies:

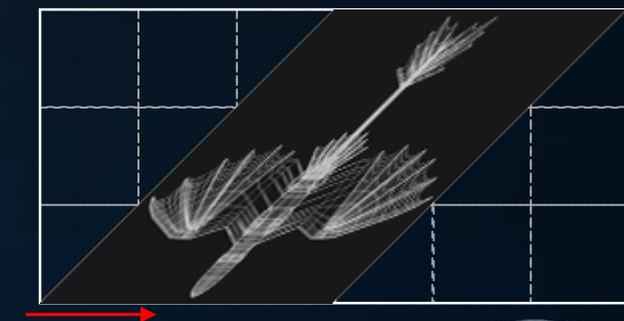
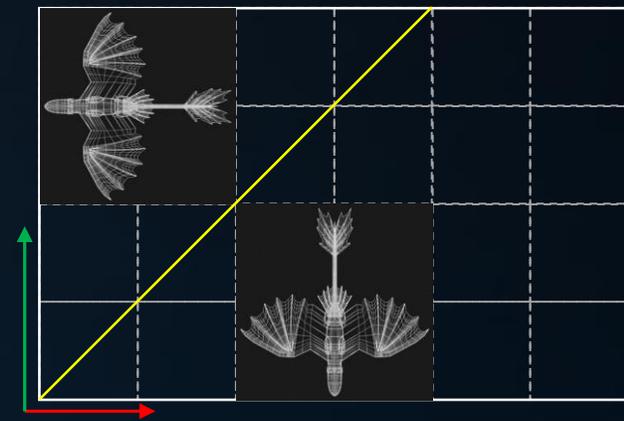
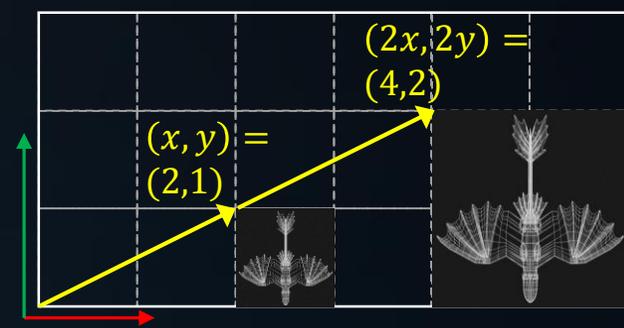
1. $T(\vec{u} + \vec{v}) = T(\vec{u}) + T(\vec{v})$
for all $\vec{u}, \vec{v} \in \mathbb{R}^n$.
2. $T(c\vec{v}) = cT(\vec{v})$
for all $\vec{v} \in \mathbb{R}^n$ and all scalars c .

Linear transformations can be represented by matrices.

We can summarize both conditions into one equation:

$$T(c_1\vec{u} + c_2\vec{v}) = c_1T(\vec{u}) + c_2T(\vec{v})$$

for all $\vec{u}, \vec{v} \in \mathbb{R}^n$ and all scalars c_1, c_2 .



Transforms

Spaces – Linear transformations

$$T(c_1\vec{u} + c_2\vec{v}) = c_1T(\vec{u}) + c_2T(\vec{v})$$

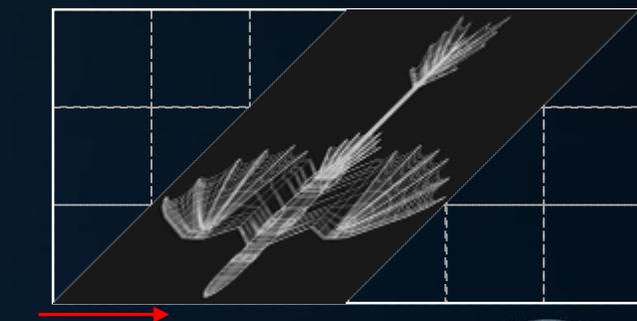
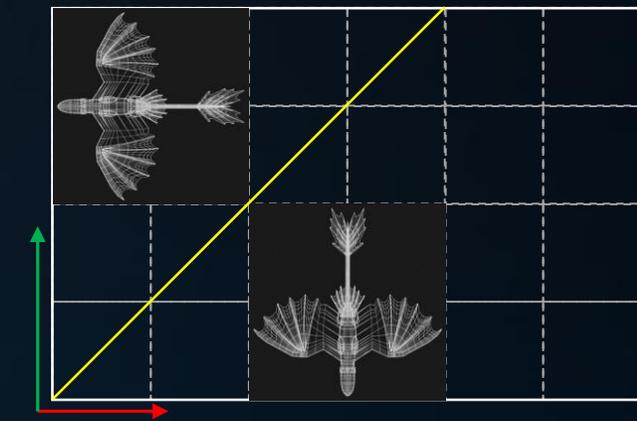
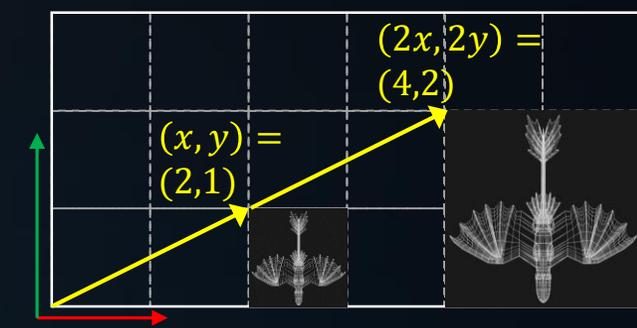
for all $\vec{u}, \vec{v} \in \mathbb{R}^n$ and all scalars c_1, c_2 .

Remember Cartesian coordinates, where each vector \vec{w} can be expressed as a linear combination of base vectors \vec{u} and \vec{v} :

$$\vec{w} = \begin{pmatrix} x \\ y \end{pmatrix} = x \begin{pmatrix} 1 \\ 0 \end{pmatrix} + y \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

If we apply a linear transform T to this vector, we get

$$T\left(\begin{pmatrix} x \\ y \end{pmatrix}\right) = T\left(x \begin{pmatrix} 1 \\ 0 \end{pmatrix} + y \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) = xT\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}\right) + yT\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}\right)$$



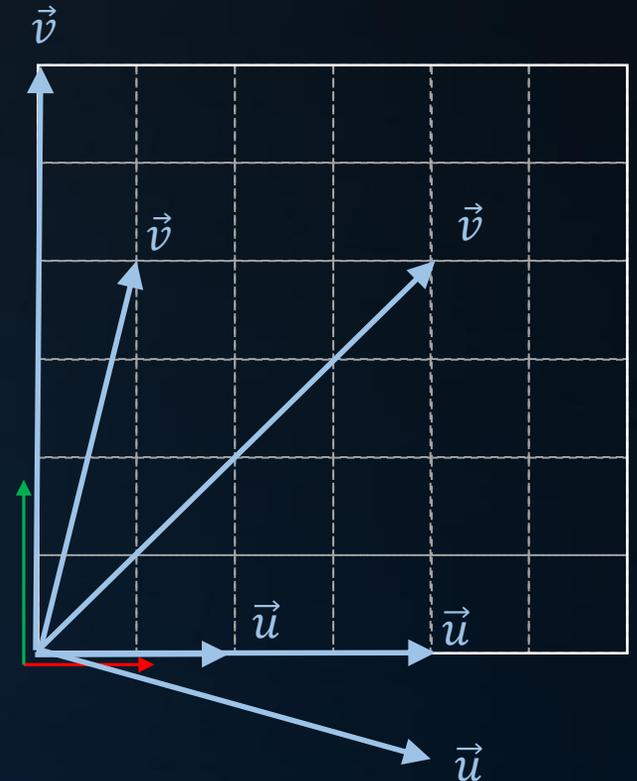
Transforms

Spaces – Linear transformations

$$T(c_1\vec{u} + c_2\vec{v}) = c_1T(\vec{u}) + c_2T(\vec{v})$$

for all $\vec{u}, \vec{v} \in \mathbb{R}^n$ and all scalars c_1, c_2 .

Matrices are constructed conveniently using two base vectors.



Transforms

Spaces – Transforming normals

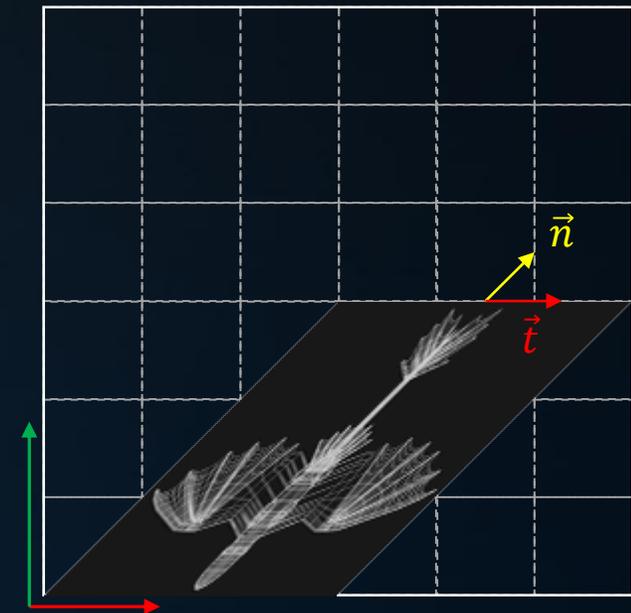
Unfortunately, normals are not always transformed correctly.

To transform a normal vector \vec{n} correctly under a given linear transformation A , we have to apply the matrix

$$(A^{-1})^T$$

Why?

Note: if the transform is orthonormal, $A^{-1} = A^T$; therefore $(A^{-1})^T = A$.



Transforms

Spaces – Transforming normals

We know that tangent vectors are transformed correctly: $A\vec{t} = \vec{t}_A$. But: $A\vec{n} \neq \vec{n}_A$.

Goal: find a matrix M that transforms \vec{n} correctly, i.e. $M\vec{n} = \vec{n}_M$, where \vec{n}_M is the correct normal of the transformed surface.

Because the original normal vector \vec{n} is perpendicular to the original tangent vector \vec{t} , we know that $\vec{n} \cdot \vec{t} = 0$. This is the same as $\vec{n} I \vec{t} = 0$. Since $I = A^{-1}A$, this is the same as $\vec{n} (A^{-1}A) \vec{t} = 0$.

Because $A\vec{t} = \vec{t}_A$ is the correctly transformed tangent vector, we have $\vec{n} A^{-1} \vec{t}_A = 0$.

Because their scalar product is 0, $\vec{n} A^{-1}$ must be orthogonal to \vec{t}_A . So, the vector we are looking for must be: $\vec{n}_M = \vec{n} A^{-1}$ (which suggests $M = A^{-1}$).

Because of how matrix multiplication is defined, \vec{n}_M and \vec{n} are transposed vectors. We can rewrite this to $\vec{n}_M = (\vec{n}^T A^{-1})^T$. And finally, remember that $(AB)^T = B^T A^T$, which gets us $\vec{n}_M = (A^{-1})^T \vec{n}$.



Transforms

Spaces – Needful things

Three things left undiscussed:

1. Reverting a transform
2. Combining transforms
3. Translation

Reverting a transform:

Invert the matrix.

Note: doesn't always work; e.g. the matrix for orthographic projection has no inverse.

Combining transforms:

Use matrix multiplication.

Note: matrix multiplication is not commutative, mind the order!

```

...
    & (depth < MAXDEPTH)
...
    t = inside / nc;
    nt = nt / nc;
    cos2t = 1.0f - nt;
    D, N );
    )
...
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0)
    Tr) R = (D * nnt - N * (a
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
...
survive = SurvivalProbability( diffuse
estimation - doing it properly, closely
if;
radiance = SampleLight( &rand, I, M, Alignment
e.x + radiance.y + radiance.z) > 0) && (depth <
...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Survival
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following death
ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    r1 = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 8: “3D Engine Fundamentals”

END of “Engine Fundamentals”

next lecture: “Transformations”

