Geometry

Informática Gráfica

Diego Gutiérrez – Adolfo Muñoz – Adrián Jarabo

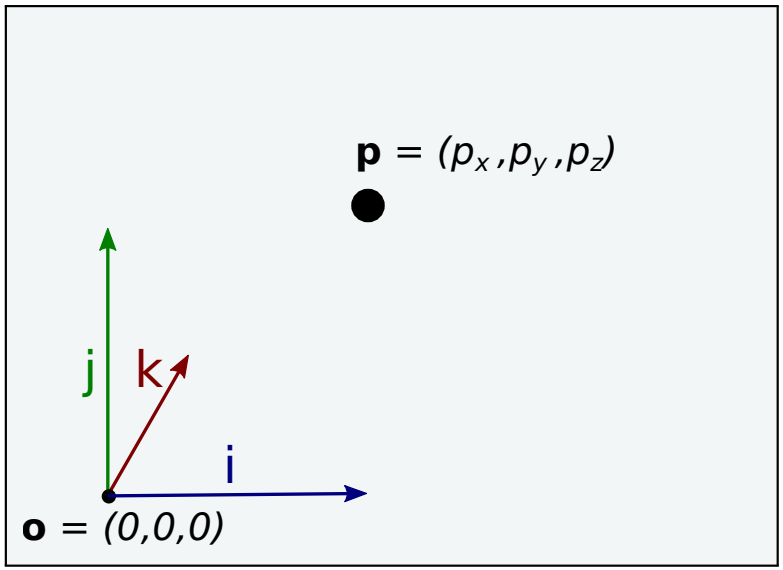
Grado en Ingeniería Informática

Graphics and Imaging Lab

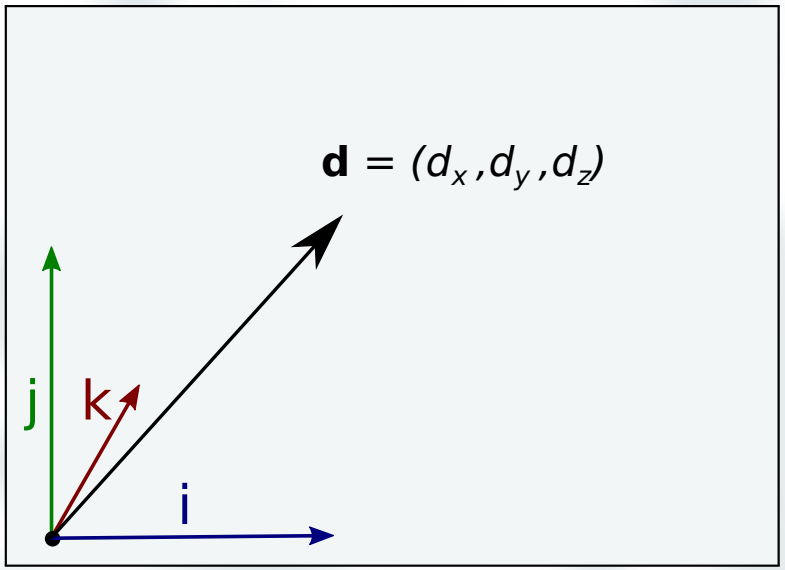
Basics

Basics

Point

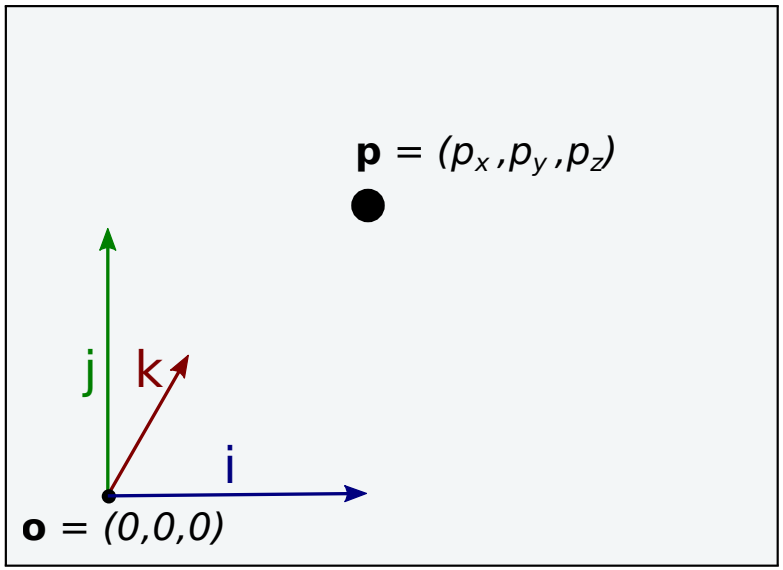


Direction



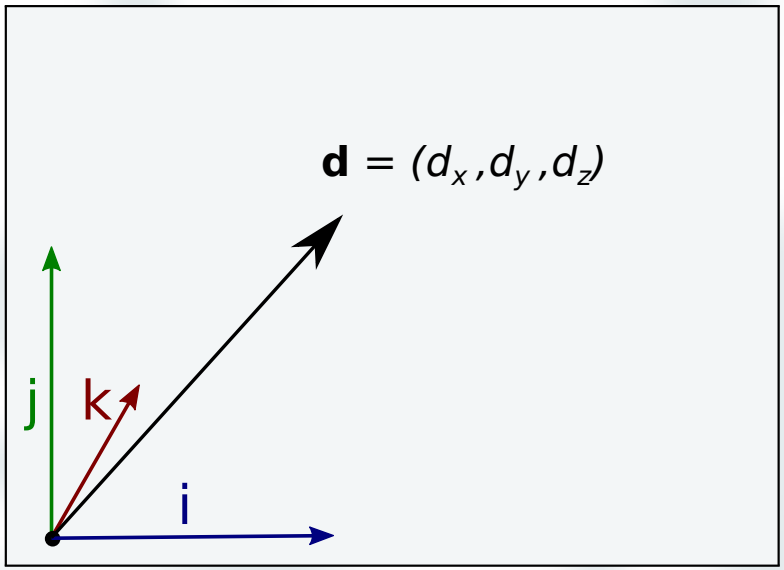
Basics

Point



$$\mathbf{p} = \mathbf{o} + p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$$

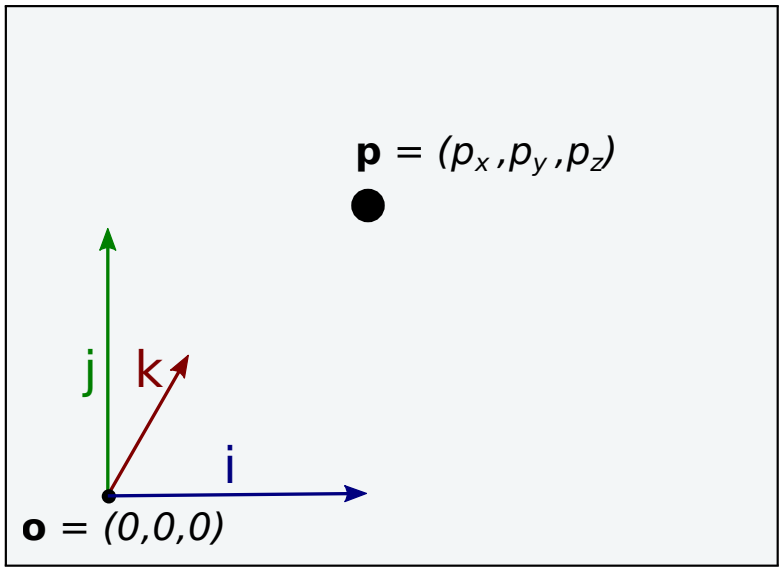
Direction



$$\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$$

Basics

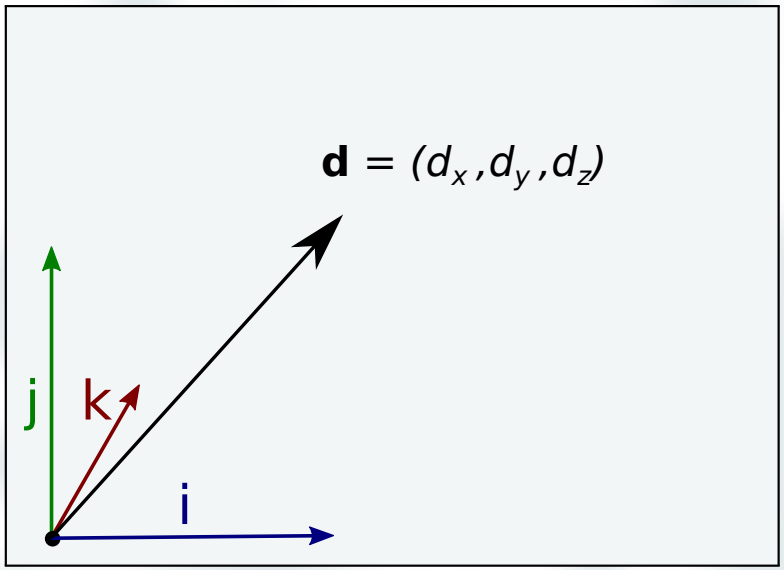
Point



$$\mathbf{p} = \mathbf{o} + p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$$

3 real numbers: p_x, p_y, p_z

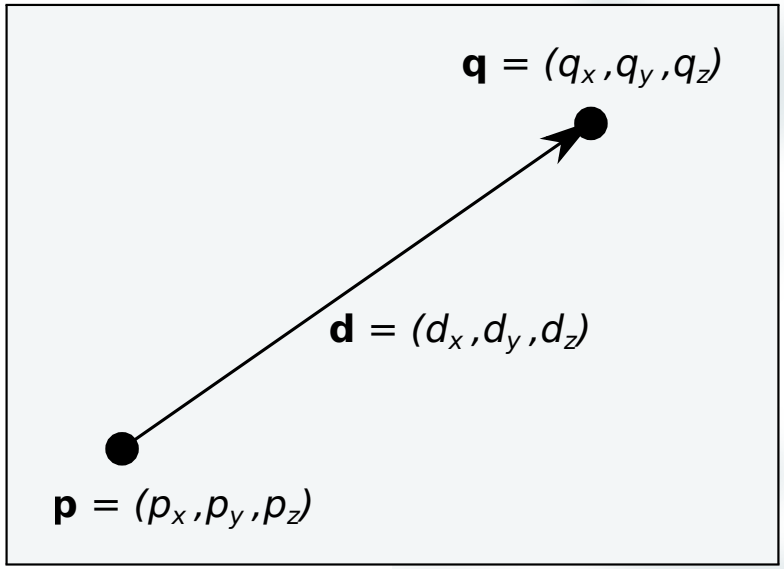
Direction



$$\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$$

3 real numbers: d_x, d_y, d_z

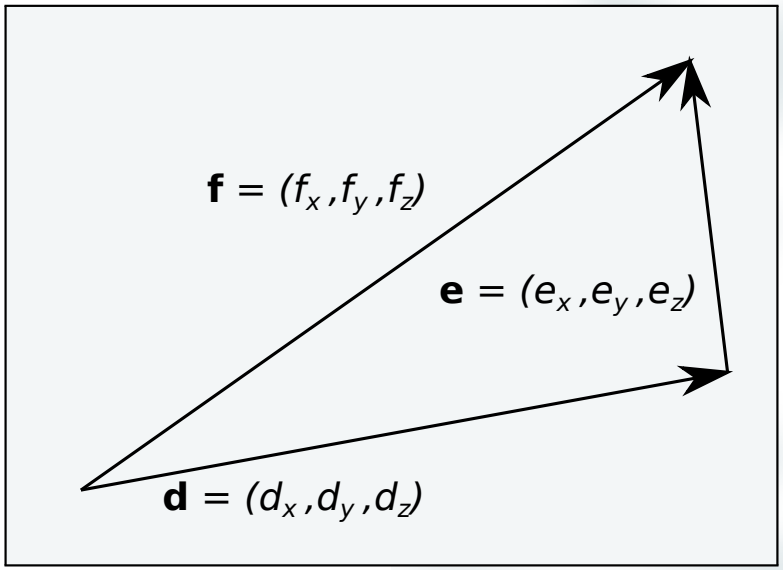
Operations



$$\mathbf{q} = \mathbf{p} + \mathbf{d} = (p_x + d_x, p_y + d_y, p_z + d_z)$$
$$\mathbf{d} = \mathbf{q} - \mathbf{p} = (q_x - p_x, q_y - p_y, q_z - p_z)$$

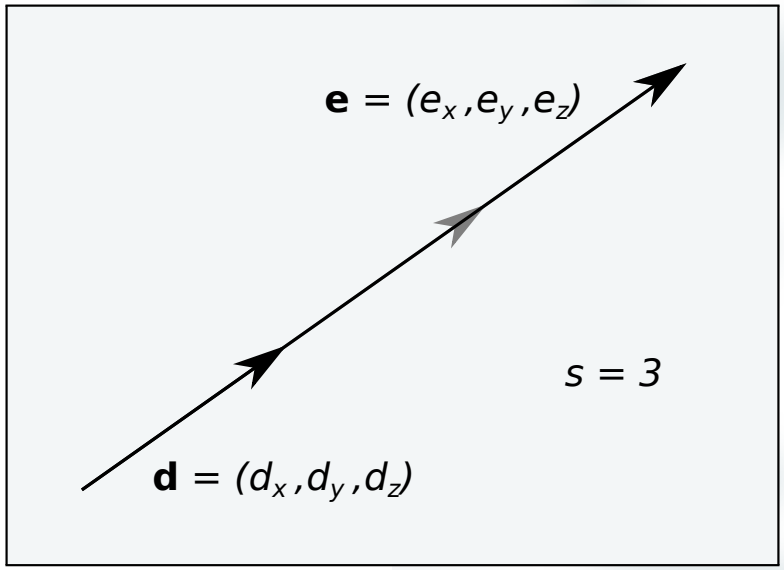
Adding two points has no geometrical meaning.

Operations



$$\mathbf{f} = \mathbf{d} + \mathbf{e} = (d_x + e_x, d_y + e_y, d_z + e_z)$$
$$\mathbf{e} = \mathbf{f} - \mathbf{d} = (f_x - d_x, f_y - d_y, f_z - d_z)$$

Operations

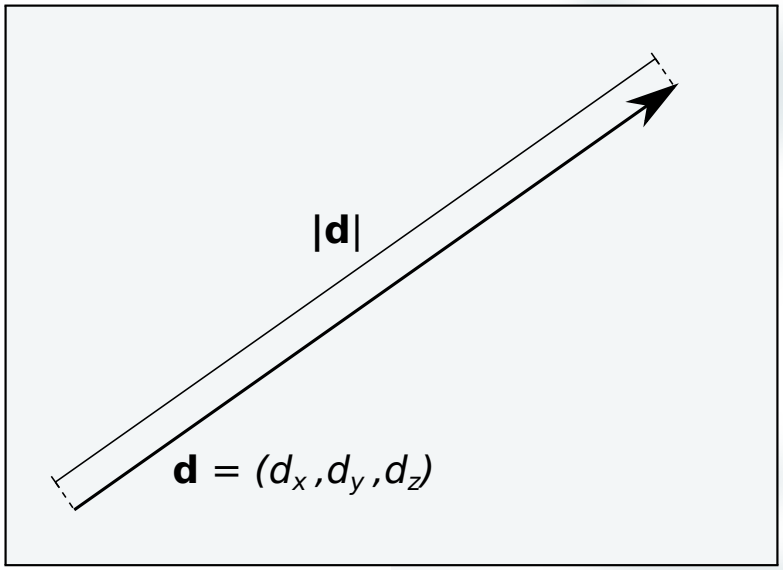


$$\mathbf{e} = s\mathbf{d} = (sd_x, sd_y, sd_z)$$

$$\mathbf{d} = \frac{\mathbf{e}}{s} = \left(\frac{e_x}{s}, \frac{e_y}{s}, \frac{e_z}{s}\right)$$

Scaling a single point has no geometrical meaning.

Modulus

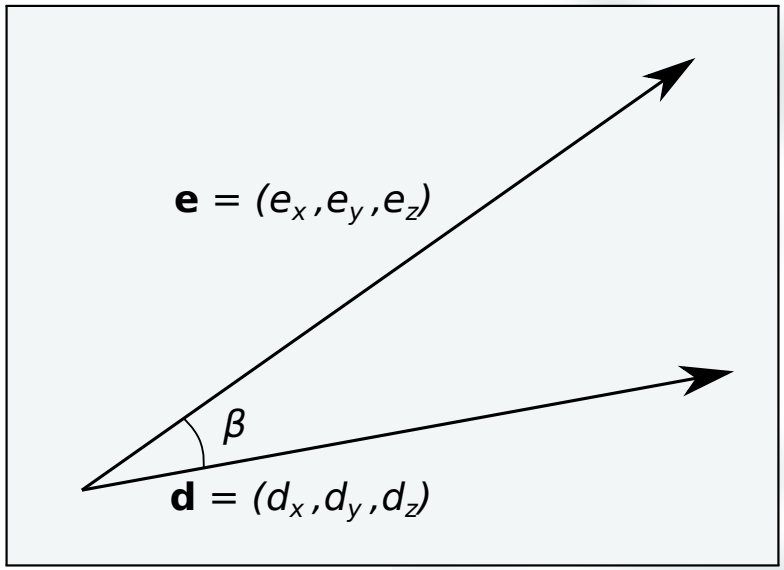


$$|\mathbf{d}| = \sqrt{d_x^2 + d_y^2 + d_z^2}$$

$$|s\mathbf{d}| = s|\mathbf{d}|$$

Modulus of a single point has no geometrical meaning.

Dot product

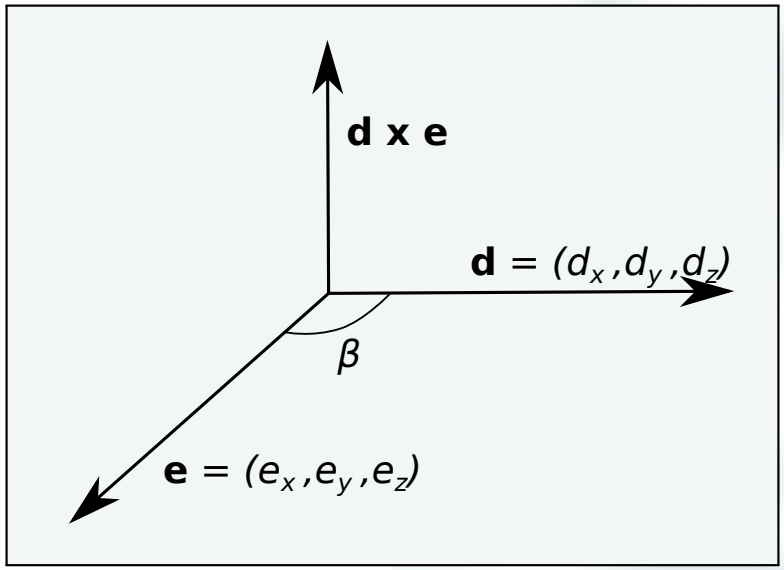


$$\mathbf{d} \cdot \mathbf{e} = d_x e_x + d_y e_y + d_z e_z \qquad \mathbf{d} \cdot \mathbf{d} = |\mathbf{d}|^2$$

$$\mathbf{d} \cdot \mathbf{e} = |\mathbf{d}||\mathbf{e}| \cos \beta \qquad \mathbf{d} \perp \mathbf{e} \Rightarrow \mathbf{d} \cdot \mathbf{e} = 0$$

Dot products with points have no geometrical sense (but may be used).

Cross product



$$|\mathbf{d} \times \mathbf{e}| = |\mathbf{d}||\mathbf{e}| \sin \beta$$

$$\mathbf{d} \times \mathbf{e} = -\mathbf{e} \times \mathbf{d}$$

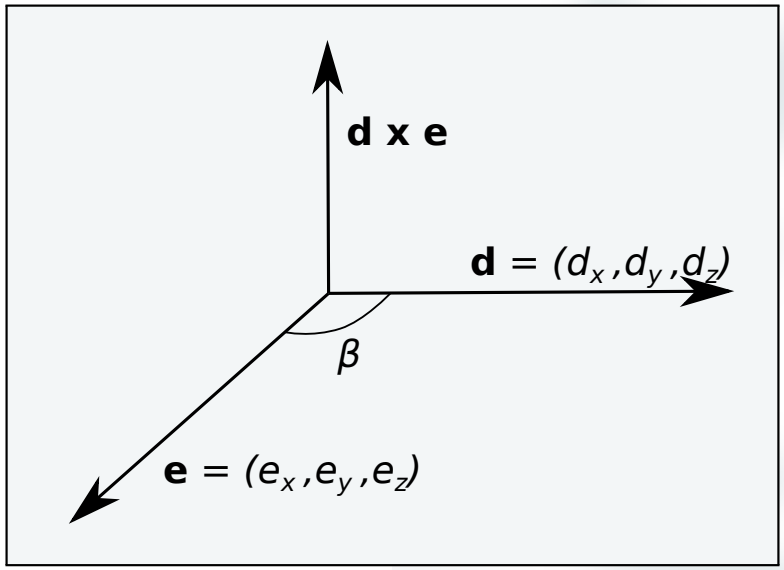
$$(\mathbf{d} \times \mathbf{e}) \perp \mathbf{d}$$

$$(\mathbf{d} \times \mathbf{e}) \perp \mathbf{e}$$

$$\mathbf{d} \times \mathbf{d} = \mathbf{0}$$

Cross products with points have no sense.

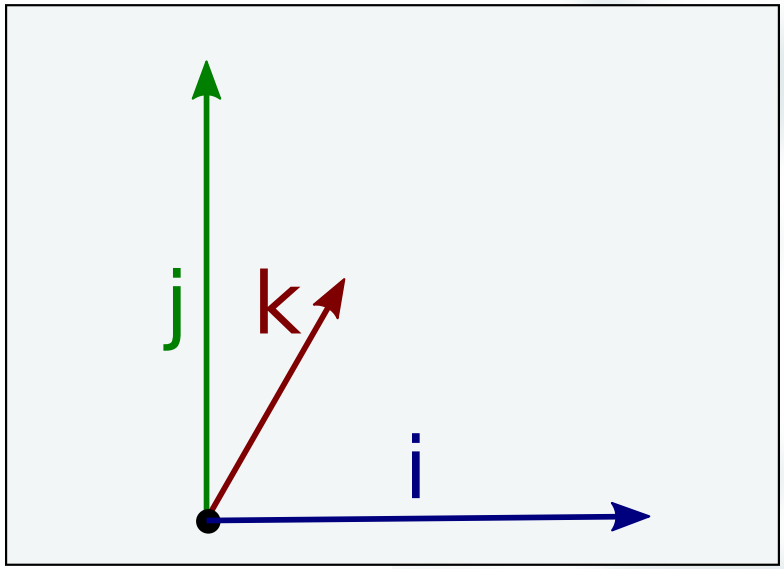
Cross product



Distributivity

$$(s_1\mathbf{d_1} + s_2\mathbf{d_2}) \times (s_3\mathbf{e_1} + s_4\mathbf{e_2}) = s_1s_3(\mathbf{d_1} \times \mathbf{e_1}) + s_2s_3(\mathbf{d_2} \times \mathbf{e_1}) + s_1s_4(\mathbf{d_1} \times \mathbf{e_2}) + s_2s_4(\mathbf{d_2} \times \mathbf{e_2})$$

Cross product



$$\mathbf{i} \times \mathbf{i} = \mathbf{0}$$
$$\mathbf{j} \times \mathbf{j} = \mathbf{0}$$
$$\mathbf{k} \times \mathbf{k} = \mathbf{0}$$

$$\mathbf{i} \times \mathbf{j} = \mathbf{k}$$
$$\mathbf{j} \times \mathbf{k} = \mathbf{i}$$
$$\mathbf{k} \times \mathbf{i} = \mathbf{j}$$

So?

Cross product

$$\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$$
$$\mathbf{e} = e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k}$$

Cross product

$$\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$$
$$\mathbf{e} = e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k}$$

$$\mathbf{d} \times \mathbf{e} = (d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}) \times (e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k})$$

Cross product

$$\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$$
$$\mathbf{e} = e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k}$$

$$\mathbf{d} \times \mathbf{e} = (d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}) \times (e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k})$$

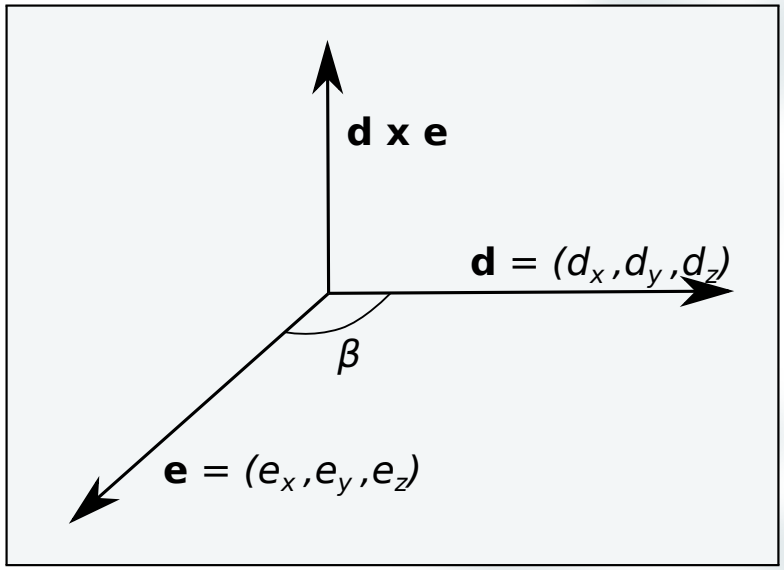
$$\begin{aligned}\mathbf{d} \times \mathbf{e} = {} & d_x e_x(\mathbf{i} \times \mathbf{i}) + d_y e_y(\mathbf{j} \times \mathbf{j}) + d_z e_z(\mathbf{k} \times \mathbf{k}) \\ & + d_x e_y(\mathbf{i} \times \mathbf{j}) + d_y e_z(\mathbf{j} \times \mathbf{k}) + d_z e_x(\mathbf{k} \times \mathbf{i}) \\ & + d_y e_x(\mathbf{j} \times \mathbf{i}) + d_z e_y(\mathbf{k} \times \mathbf{j}) + d_x e_z(\mathbf{i} \times \mathbf{k})\end{aligned}$$

Cross product

$$\begin{aligned}\mathbf{d} \times \mathbf{e} = {} & d_x e_x (\mathbf{i} \times \mathbf{i}) + d_y e_y (\mathbf{j} \times \mathbf{j}) + d_z e_z (\mathbf{k} \times \mathbf{k}) \\ & + d_x e_y (\mathbf{i} \times \mathbf{j}) + d_y e_z (\mathbf{j} \times \mathbf{k}) + d_z e_x (\mathbf{k} \times \mathbf{i}) \\ & + d_y e_x (\mathbf{j} \times \mathbf{i}) + d_z e_y (\mathbf{k} \times \mathbf{j}) + d_x e_z (\mathbf{i} \times \mathbf{k})\end{aligned}$$

$$\begin{aligned}\mathbf{d} \times \mathbf{e} = {} & d_x e_y \mathbf{k} + d_y e_z \mathbf{i} + d_z e_x \mathbf{j} \\ & - d_y e_x \mathbf{k} - d_z e_y \mathbf{i} - d_x e_z \mathbf{j} \\ = {} & (d_y e_z - d_z e_y)\mathbf{i} + (d_z e_x - d_x e_z)\mathbf{j} + (d_x e_y - d_y e_x)\mathbf{k}\end{aligned}$$

Cross product



$$\mathbf{d} \times \mathbf{e} = (d_y e_z - d_z e_y)\mathbf{i} + (d_z e_x - d_x e_z)\mathbf{j} + (d_x e_y - d_y e_x)\mathbf{k}$$

Operations overview

- Point represented as $\mathbf{p}$.
- Direction represented as $\mathbf{d}$.
- Scalar represented as s.

$$\mathbf{d} \cdot \mathbf{d} = s$$
$$\mathbf{p} + \mathbf{d} = \mathbf{p}$$
$$\mathbf{d} + \mathbf{d} = \mathbf{d}$$
$$s\mathbf{d} = \mathbf{d}$$

$$\mathbf{d} \times \mathbf{d} = \mathbf{d}$$
$$\mathbf{p} - \mathbf{p} = \mathbf{d}$$
$$\mathbf{d} - \mathbf{d} = \mathbf{d}$$
$$\frac{\mathbf{d}}{s} = \mathbf{d}$$

Implementation details

Remember some **advice**:

- Prefer **functional** over state machine behavior.
- Prefer **operators** over long function / method names.
- Be effective, do not **overdesign**.
- Avoid **memory management**. It is not your battle (this time).
- **Premature optimization** is the root of all evil (Donald Knuth).
- Choose the **right programming language** for you.
- **Enjoy** visualizing your results.

Implementation details

- Be effective, do not **overdesign**.

Question : Do you need separate data types for points and directions (and RGB tuples)?

Implementation details

- Be effective, do not **overdesign**.

Question : Do you need separate data types for points and directions (and RGB tuples)?
Answer : Entirelly up to you.

Implementation details

- Be effective, do not **overdesign**.

Question : Do you need separate data types for points and directions (and RGB tuples)?
Answer : Entirelly up to you.

Pros

- Specific behavior
- Compile time checks

Cons

- Lots of common behavior
- Extra code

Implementation details

- Avoid **memory management**. It is not your battle (this time).

Question : Which data types represent the three coordinates?

Implementation details

- Avoid **memory management**. It is not your battle (this time).

Question : Which data types represent the three coordinates?
Answer : Anything that avoids memory creation / destruction.

Implementation details

- Avoid **memory management**. It is not your battle (this time).

Java

Right or wrong?

```
1  class Point {
2      float[] c;
3      public Point(float x, float y, float z) {
4          c = new float[3];
5          c[0]=x; c[1]=y; c[2]=z;
6      }
7  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

Java

Right or wrong?

```
1  class Point {
2      float[] c;
3      public Point(float x, float y, float z) {
4          c = new float[3];
5          c[0]=x; c[1]=y; c[2]=z;
6      }
7  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

Java

Right or wrong?

```
1  class Point {
2      float cx, cy, cz;
3      public Point(float x, float y, float z) {
4          cx=x; cy=y; cz=z;
5      }
6  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

Java

Right or wrong?

```
1  class Point {
2      float cx, cy, cz;
3      public Point(float x, float y, float z) {
4          cx=x; cy=y; cz=z;
5      }
6  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

C++

Right or wrong?

```
1  class Point {
2      std::vector<float> c;
3      public Point(float x, float y, float z) : c(3) {
4          c[0]=x; c[1]=y; c[2]=z;
5      }
6  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

C++

Right or wrong?

```
1  class Point {
2      std::vector<float> c;
3      public Point(float x, float y, float z) : c(3) {
4          c[0]=x; c[1]=y; c[2]=z;
5      }
6  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

C++

Right or wrong?

```
1  class Point {
2     float c[3]; //or std::array<float,3> c;
3     public Point(float x, float y, float z) {
4        c[0]=x; c[1]=y; c[2]=z;
5     }
6  };
```

Implementation details

- Avoid **memory management**. It is not your battle (this time).

C++

Right or wrong?

```
1  class Point {
2     float c[3]; //or std::array<float,3> c;
3     public Point(float x, float y, float z) {
4        c[0]=x; c[1]=y; c[2]=z;
5     }
6  };
```

Implementation details

- Be effective, do not **overdesign**.
- Avoid **memory management**. It is not your battle (this time).

C++

Right or wrong?

```
1  template<typename real, unsigned int N>
2  class Point {
3      real c[N];
4      public Point(real x, real y, real z) {
5          c[0]=x; c[1]=y; c[2]=z;
6      }
7  };
```

Implementation details

- Be effective, do not **overdesign**.
- Avoid **memory management**. It is not your battle (this time).

C++

Right or wrong?

```
1  template<typename real, unsigned int N>
2  class Point {
3      real c[N];
4      public Point(real x, real y, real z) {
5          c[0]=x; c[1]=y; c[2]=z;
6      }
7  };
```

It depends. It is overkill in your case.

Transformations

Homogeneous coordinates

Goal: Find a compact and homogeneous representation of 3D geometrical transformations.

Why?

- More reusable code.
- Use specific hardware (aka. GPUs).

Homogeneous coordinates

Goal: Find a compact and homogeneous representation of 3D geometrical transformations.

Problem

Points and **directions** behave differently with respect to the same transformation.

Example: In a translation, a point should move but a direction should not.

Homogeneous coordinates

Goal: Find a compact and homogeneous representation of 3D geometrical transformations.

Solution

Homogeneous coordinates: Add a fourth cordinate that separates points and directions. A direction is considered to be ”a point in infinity”.

Homogeneous coordinates

Given a 4×4 transform matrix $\mathbf{T}$:

- Take the 4×1 vector $\mathbf{v}$ that represents the point $\mathbf{p}$ or direction $\mathbf{d}$ with the homogeneous coordinate.
- Calculate the product $\mathbf{v}' = \mathbf{T}\mathbf{v}$.
- Take the 4×1 transformed vector $\mathbf{v}'$ and get the corresponding point $\mathbf{p}'$ or direction $\mathbf{d}'$.

Homogeneous coordinates

Point

$$\mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

$$\mathbf{v} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

Direction

$$\mathbf{d} = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}$$

$$\mathbf{v} = \begin{pmatrix} d_x \\ d_y \\ d_z \\ 0 \end{pmatrix}$$

Homogeneous coordinates

Point

$$\mathbf{v}' = \begin{pmatrix} v'_x \\ v'_y \\ v'_z \\ v'_w \end{pmatrix}$$

$$\mathbf{p}' = \begin{pmatrix} \frac{v'_x}{v'_w} \\ \frac{v'_y}{v'_w} \\ \frac{v'_z}{v'_w} \end{pmatrix}$$

What if $v'_w = 0$?.

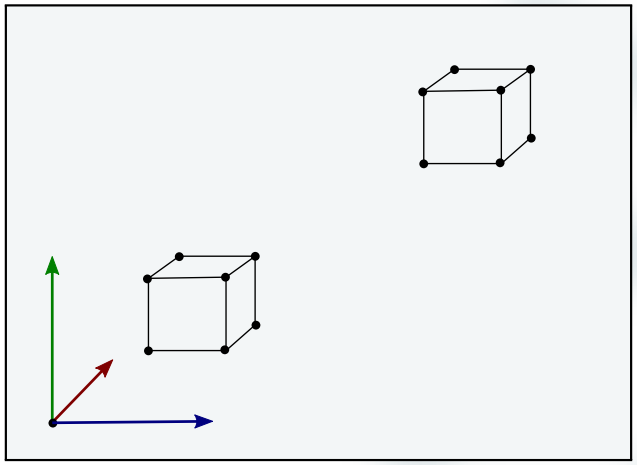
Direction

$$\mathbf{v}' = \begin{pmatrix} v'_x \\ v'_y \\ v'_z \\ v'_w \end{pmatrix}$$

$$\mathbf{d}' = \begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix}$$

What if $v'_w \neq 0$?.

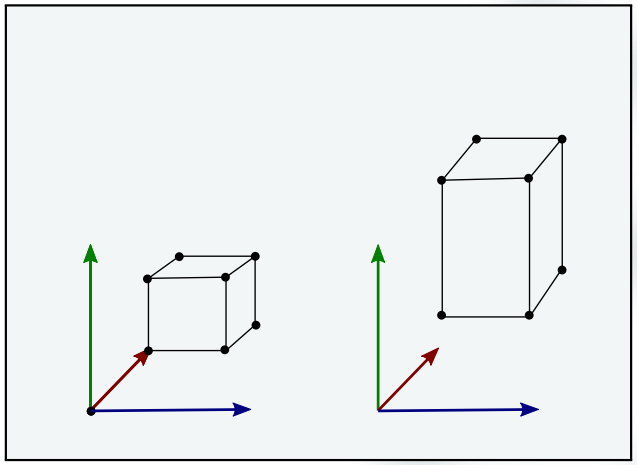
Translation



Displace along the three axes (t_x, t_y, t_z)

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

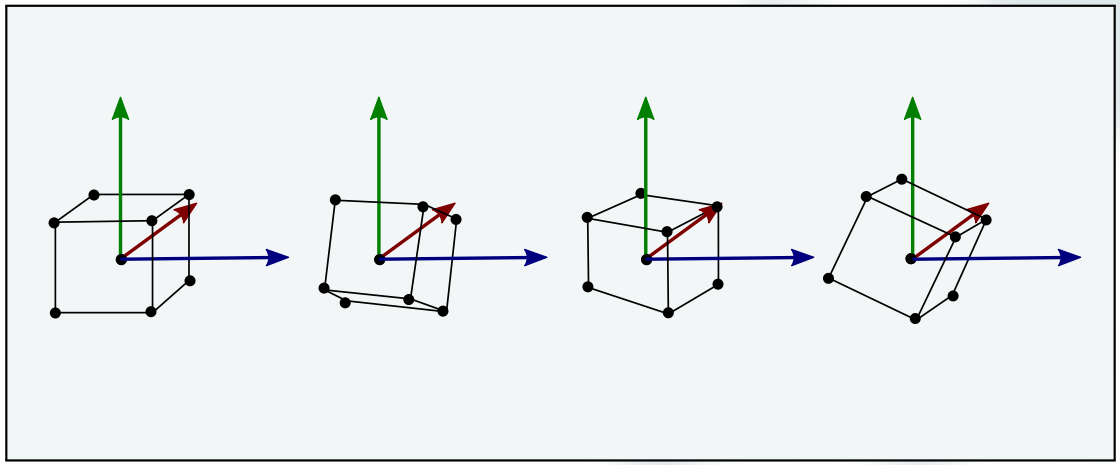
Scale



Scale the object by a factor on each axes (s_x, s_y, s_z)

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation



Rotate an angle θ along an axis.

X axis

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

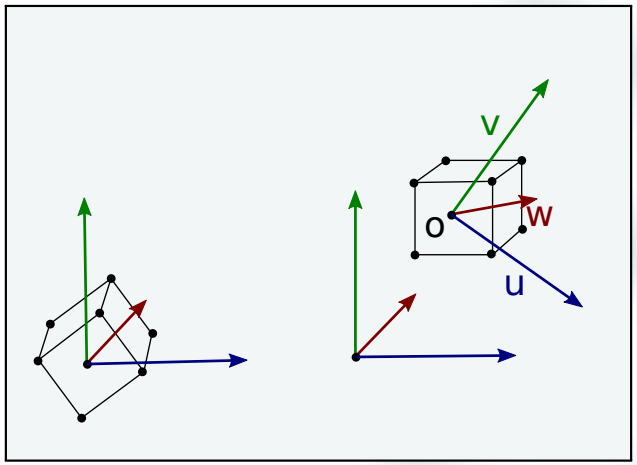
Y axis

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Z axis

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Change of base



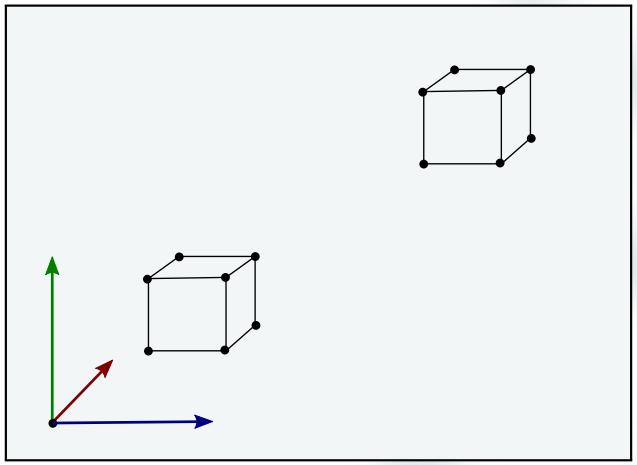
Move to a new base with origin in $\mathbf{o}$ and axes in $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$

$$\begin{pmatrix} u_x & v_x & w_x & o_x \\ u_y & v_y & w_y & o_y \\ u_z & v_z & w_z & o_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse transform

The **inverse transform matrix** of a transform matrix undoes what the original transform matrix does.

Inverse transform

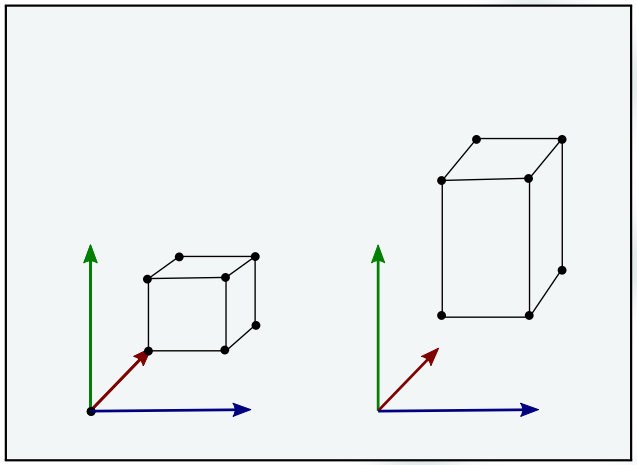


Task

Calculate the inverse transform matrix of a translation.

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse transform



Task

Calculate the inverse transform matrix of a scale.

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

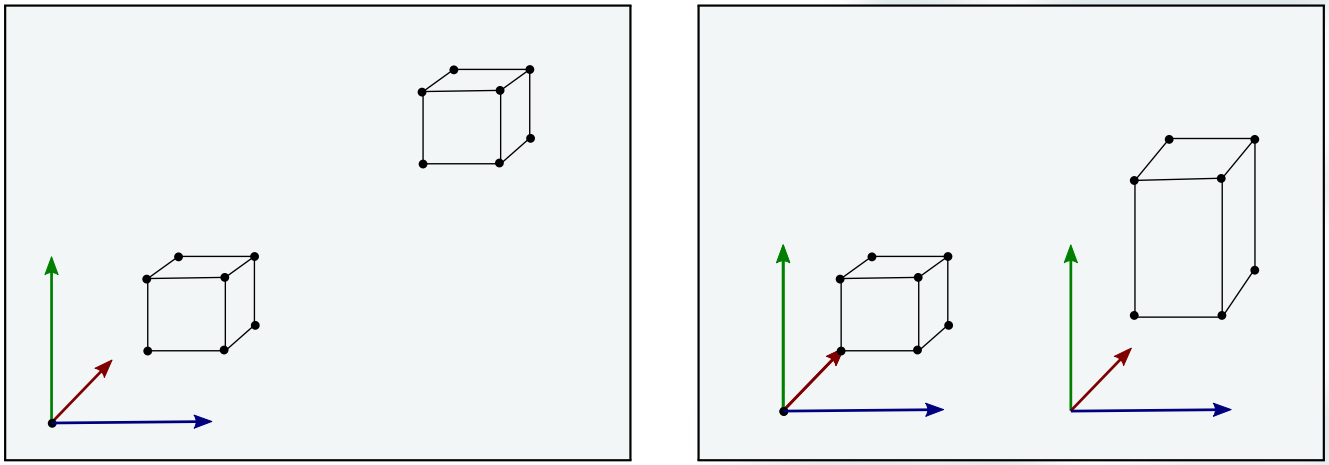
Inverse transform

Task

- Matrix product: translation by inverse translation.
- Matrix product: scale by inverse scale.

What can you conclude?

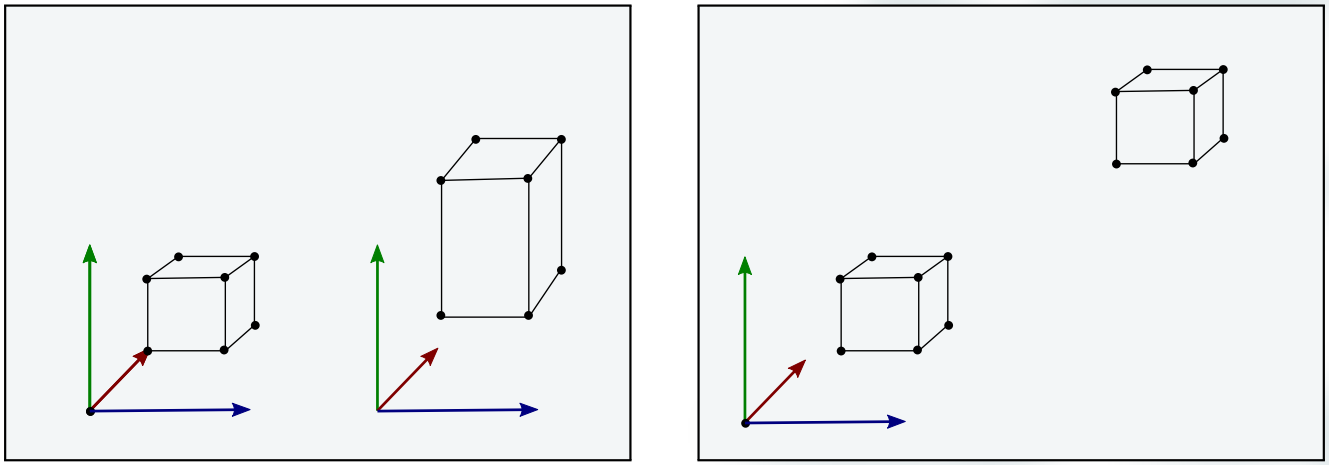
Combine transforms



Task

Calculate the matrix of the combined transform: first a translation, then a scale. How?

Combine transforms



Task

Calculate the matrix of the combined transform: first a scale, then a translation. How?

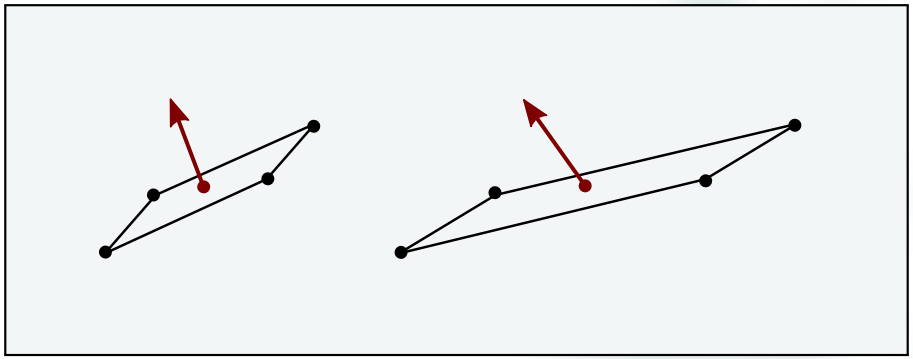
Combine transforms

Questions

- Is the application of transformations reversible? Is "translation followed by scale" the same than "scale followed by translation"?
- Is the matrix product conmutative?

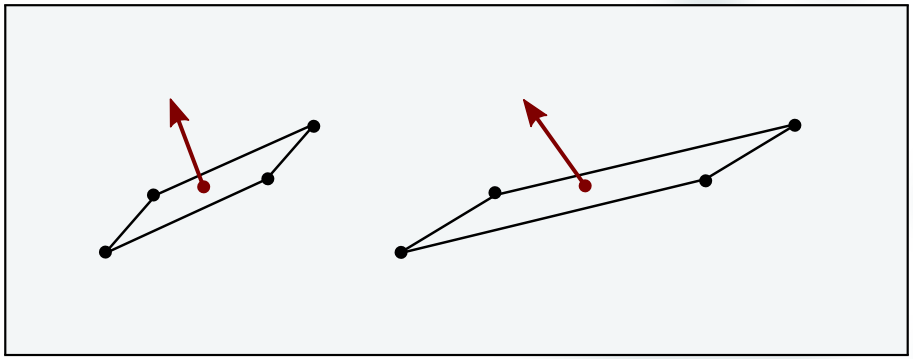
Normal directions

What happens if we transform normal directions using the same transformation matrix?

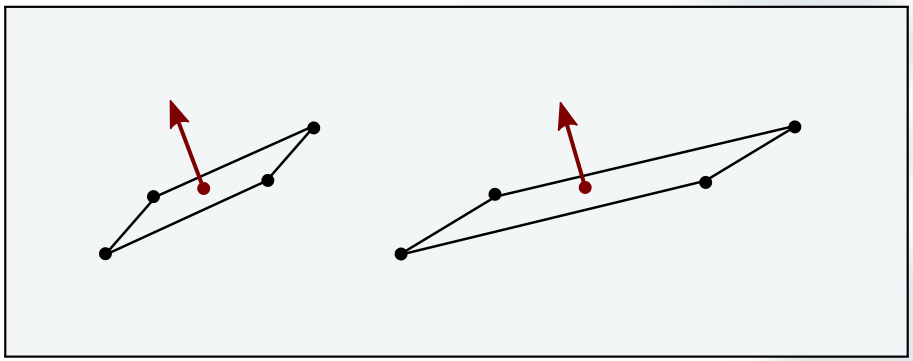


Normal directions

What happens if we transform normal directions using the same transformation matrix?



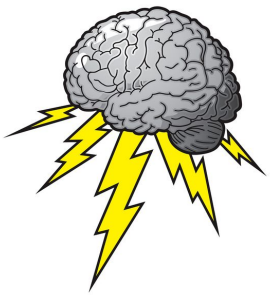
Given a transform matrix $\mathbf{M}$, normals are transformed with $(\mathbf{M}^1)^T$:



Geometrical objects

Geometrical objects

- We are able to represent points and directions in 3D space.
- We are able to transform them.



How do we represent **solid objects**?

Representations

Implicit

An implicit equation is defined for all points $\mathbf{p}$ in space

$$f(\mathbf{p}) \leq 0$$

that yields true for all points belonging to the object.
The surface of the object is therefore identified by:

$$f(\mathbf{p}) = 0$$

Parametric

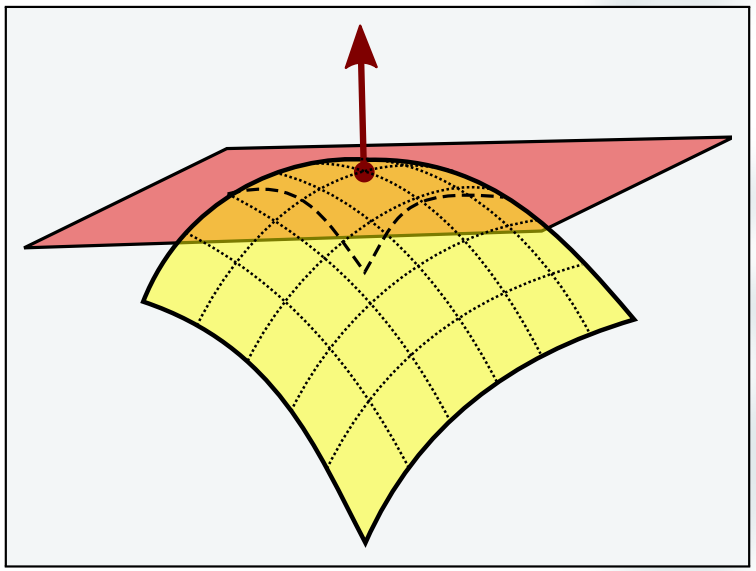
All points on the surface are defined through a parametric 3D function

$$\mathbf{p} = \mathbf{f}(u, v)$$

u and *v* are the two parameters (bounded or unbounded)
The inside and outside of the object are unclear (w.r.t. surface normal?).

Representations

The **surface normal** at a specific point of a surface *p* is the vector that is perpendicular to that surface (specifically, to its *tangent plane*).



Representations

The **surface normal** at p is perpendicular to that surface.

Implicit

$$f(\mathbf{p}) = f(p_x, p_y, p_z) = 0$$

Surface normal is:

$$n(\mathbf{p}) = \left(\frac{\partial f}{\partial p_x}(\mathbf{p}), \frac{\partial f}{\partial p_y}(\mathbf{p}), \frac{\partial f}{\partial p_z}(\mathbf{p}) \right)$$

Watch out for singular points.

Parametric

$$\mathbf{p} = \mathbf{f}(u, v)$$

Normal perpendicular to tangents.

$$\mathbf{t_u}(u, v) = \frac{\partial \mathbf{f}}{\partial u}(u, v)$$

$$\mathbf{t_v}(u, v) = \frac{\partial \mathbf{f}}{\partial v}(u, v)$$

$$\mathbf{n}(u, v) = \mathbf{t_u}(u, v) \times \mathbf{t_v}(u, v)$$

Representations

Which representation is best?

- Implicit
- Parametric

Representations

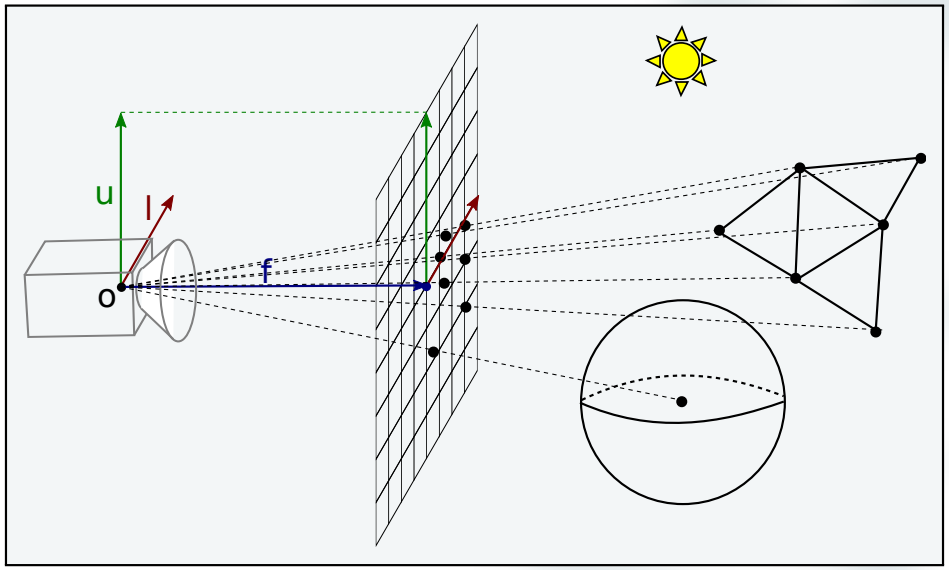
Which representation is best?

- Implicit
- Parametric

Depends on the algorithm you are going to use:

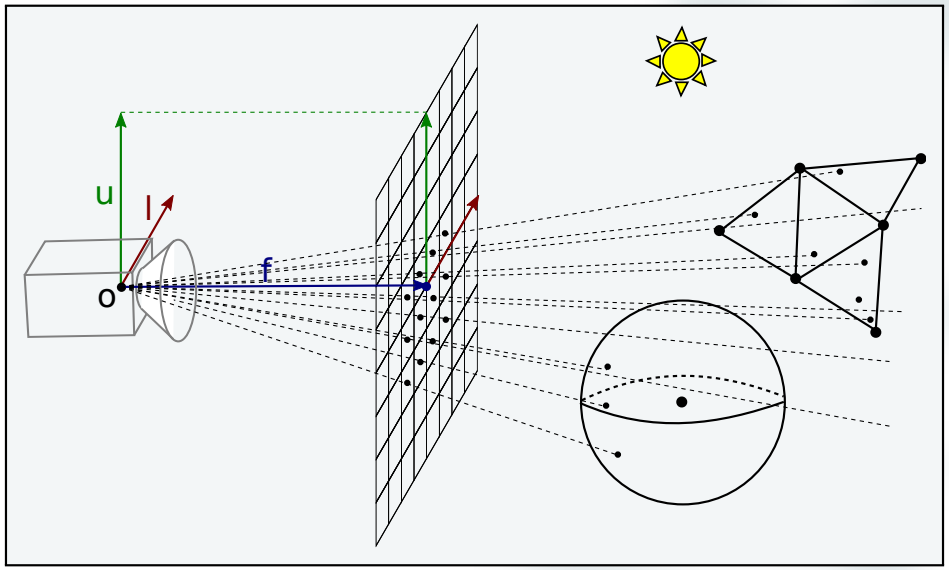
- **Rasterization:** project points to camera space and connect edges in camera space.
- **Ray tracing:** project rays from the camera and check intersections with the geometry. Better with **implicit surfaces** through system of equations.

Representations



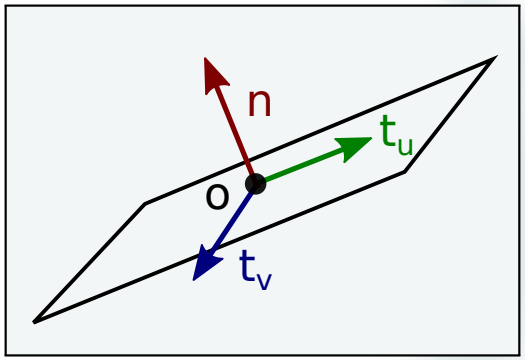
A **parametric surface** can be sampled (discretized) at certain intervals (tessellation) and generate strips of triangles.

Representations



For computing intersections, the equation of the ray is combined with the equation of the **implicit surface** to obtain a system of equations that, when solved, yields the intersection point.

Plane



Implicit

$$\mathbf{n} \cdot \mathbf{p} + d = 0$$

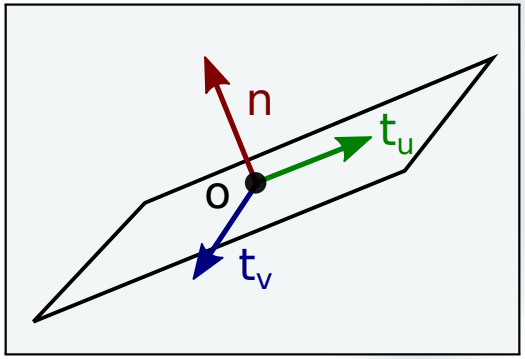
- $\mathbf{n}$ Surface normal.
- d Distance to origin.

Parametric

$$\mathbf{p}(u, v) = \mathbf{o} + u\mathbf{t_u} + v\mathbf{t_v}$$

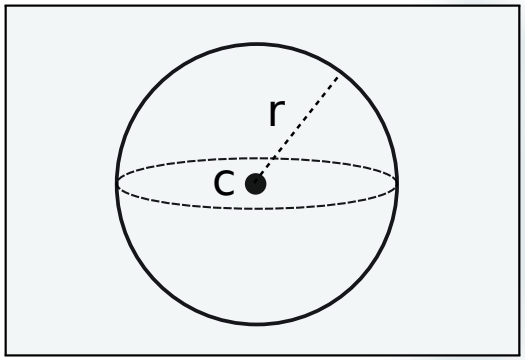
- $\mathbf{o}$ Origin
- $\mathbf{t_u}$ and $\mathbf{t_v}$ tangents.
- u and v are unbounded.

Plane



How are $\mathbf{n}$, $\mathbf{t_u}$, $\mathbf{t_v}$, $\mathbf{o}$ and d related?

Sphere



Implicit

$$(\mathbf{p}-\mathbf{c})\cdot(\mathbf{p}-\mathbf{c})-r^2=0$$

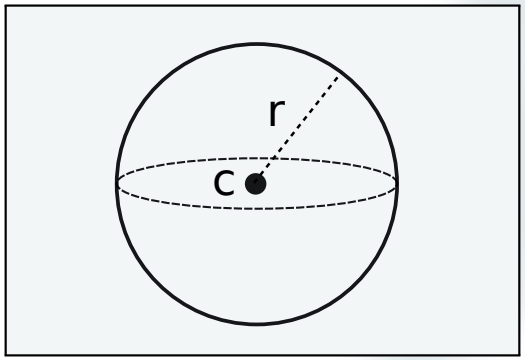
- $\mathbf{c}$ Center.
- r Radius.

Parametric

$$\mathbf{p}(\theta,\phi)=\mathbf{c}+r\begin{pmatrix}\sin\theta\sin\phi\\ \sin\theta\cos\phi\\ \cos\theta\end{pmatrix}$$

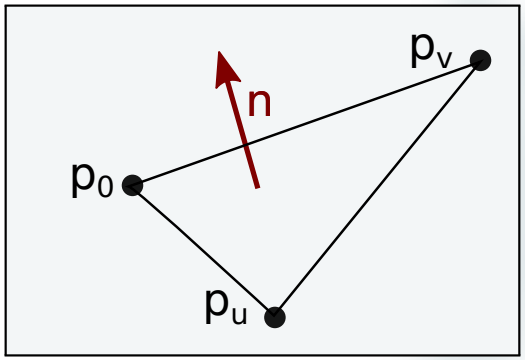
- $\mathbf{c}$ Center.
- r Radius.

Sphere



What is the geometrical meaning of the parameters θ and ϕ?

Triangle



Implicit

$$\mathbf{n} \cdot \mathbf{p} + d = 0$$

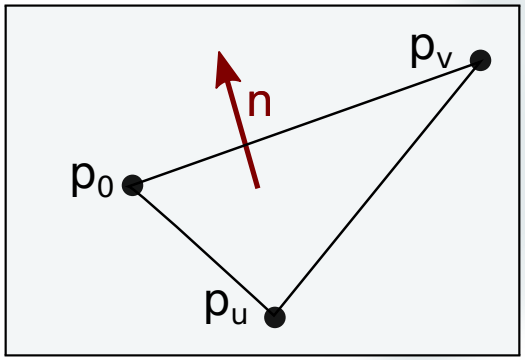
- $\mathbf{n}$ Surface normal.
- d Distance to origin.
- It's a plane? It's a bird?
- It's definitelly a plane.

Parametric

$$\mathbf{p}(u, v) = \mathbf{p_0} + u(\mathbf{p_u} - \mathbf{p_0}) + v(\mathbf{p_v} - \mathbf{p_0})$$

- $\mathbf{p_0}$, $\mathbf{p_u}$ and $\mathbf{p_v}$ vertices of the triangle.
- $u \geq 0, v \geq 0$ and $u + v \leq 1$

Triangle



How to calculate the implicit equation from vertex positions?

$$\mathbf{n} \cdot \mathbf{p} + d = 0$$

Geometry

Informática Gráfica

Diego Gutiérrez – Adolfo Muñoz – Adrián Jarabo

Grado en Ingeniería Informática

Graphics and Imaging Lab