Further Graphics

Simon

September 18, 2025

Contents

1 Geometry Representations

1.1 Considerations

When deciding on which representation to use there are a few considerations that are prevalent.

- **Storage**
- Ease of **Acquisition**
- Ease of **Creation** in software
- Ease of **Editing** features
- Ease of **Rendering**

1.2 Parametric Curves & Surfaces

Definition 1.1: Parametric Curves & Surfaces

A parametric object is one which has the general equation below for $m, n \in \mathbb{N}$

$$f : X \to Y$$

$$X \subseteq \mathbb{R}^m,\ Y \subseteq \mathbb{R}^n$$

1.2.1 Parametric Curves

Definition 1.2: Parametric Curves

Parametric Curves are when $m = 1$ and $n \in \mathbb{N} \ \cap \ n > 1$.

$$f : X \to Y$$
$$X \subseteq \mathbb{R},\ Y \subseteq \mathbb{R}^n$$

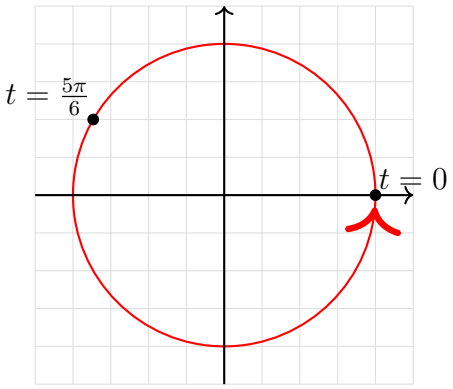
Planar Curves have n as 2, and similarly curves in 3D have n as 3.

Planar Curves have have two functions one for each basis (or axis) $x(t)$ and $y(t)$ for a parameter t

$$f : X \to Y, \quad X \subseteq \mathbb{R},\ Y \subseteq \mathbb{R}^2$$

$$\mathbf{p}(x(t),\ y(t))$$

For example a circle can be described with $x(t) = r\cos t$ and $y(t) = r\sin t$ where t is the parameter, $t \in [0,\ 2\pi)$ and r is the radius of the circle.



To define a curve in 3D we need 3 functions, one for each axis.

$$f : X \to Y, \quad X \subseteq \mathbb{R},\ Y \subseteq \mathbb{R}^3$$

$$\mathbf{p}(t) = (x(t),\ y(t),\ z(t))$$

1.2.2 Parametric Surfaces

Definition 1.3: Parametric Surfaces

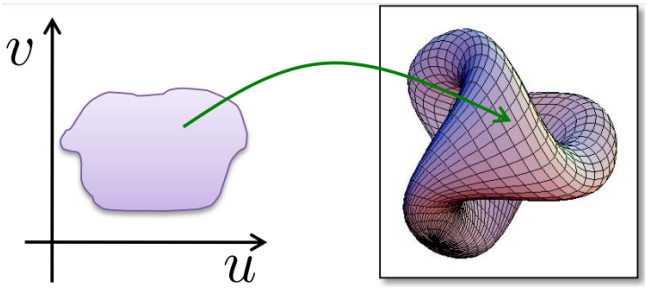
A **Parametric Surface** is a surface in $\mathbb{R}^3$ which is defined by a parametric equation with two parameters typically u and v, and from the general form $m = 2, n = 3$.

$$f : X \to Y, \quad X \subseteq \mathbb{R}^2,\ Y \subseteq \mathbb{R}^3$$

$$\mathbf{p}(u, v) = (x(u, v),\ y(u, v),\ z(u, v))$$

By setting $m = 2$ there are now 2 degrees of freedom, which allows us to define a surface. Denoting the parameter space with u and v (known as uv-space), each point on the surface is given by the functions x, y, z.

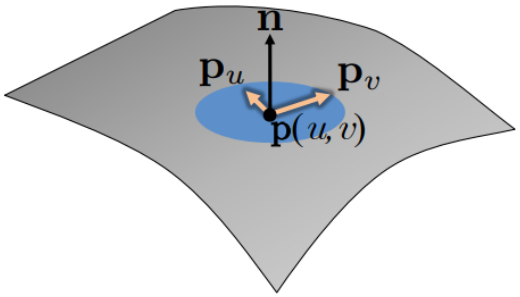
$$\mathbf{p}(u, v) = (x(u, v), y(u, v), z(u, v))$$



Locally a surface is a plane which means there are only 2 orthogonal directions you can go to still stay on the surface and the other orthogonal direction will take you either inside or outside the surface, so basically moving in the v-direction will move you in one tangent direction and moving in the u-direction will move you in the other.

The tangent plane for a point contains the vectors computed by taking the derivatives w.r.t u and v as previously stated that moving in one either u or v is a tangent direction so the partial derivative of both will be in the tangent plane.

The surface normal $\mathbf{n}$ is then the vector orthogonal to the tangent plane.



$$\mathbf{p}_u = \frac{\partial \mathbf{p}(u, v)}{\partial u},\ \mathbf{p}_v = \frac{\partial \mathbf{p}(u, v)}{\partial v}$$

Regular parametrization:

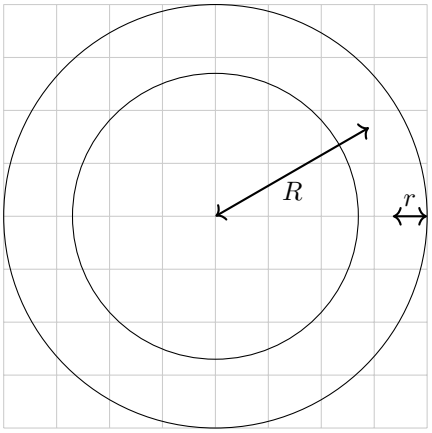
$$\mathbf{p}_u \times \mathbf{p}_v \neq \underline{0}$$

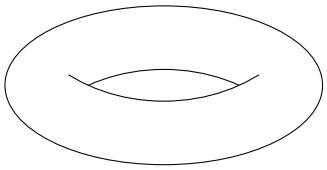
$$\hat{\mathbf{n}}(u, v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{||\mathbf{p}_u \times \mathbf{p}_v||}$$

1.2.3 Examples of Parametric Objects

3D examples:
A Torus with parameters u and v, R being the distance from the center of the tube to the center of the entire object, and r being the radius of the tube

$$x = r \sin v$$
$$y = (R + r \cos v) \sin u$$
$$z = (R + r \cos v) \cos u$$



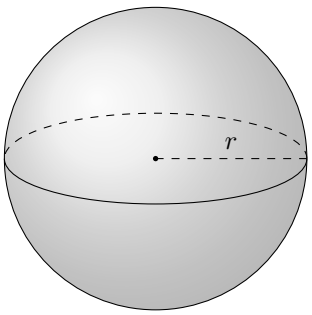


For a sphere with parameters u and v, r as the radius

$$\mathbf{p} : \mathbb{R}^2 \to \mathbb{R}^3$$

$$x = r \cos(u) \cos(v)$$
$$y = r \sin(u) \cos(v)$$
$$z = r \sin(v)$$

$$(u, v) \in [0, 2\pi) \times [-\frac{\pi}{2}, \frac{\pi}{2}]$$



1.2.4 Bezier Curves

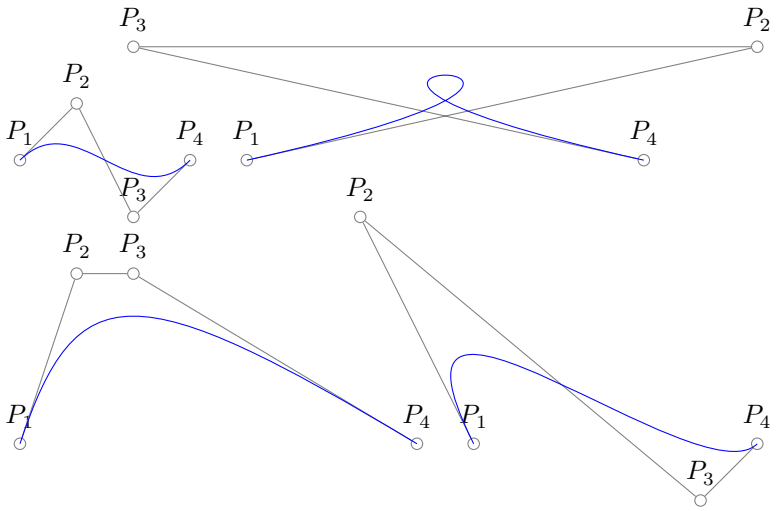
A more complex case of parametric curves is a Bezier curve.

Definition 1.4: Bezier Curves

A **Bezier Curve** is a parametric curve which has a weighted combination of basis functions (B_i^n). The weights (also called control points) are a set of points ($\mathbf{p}_i$) in 2D space.

$$\mathbf{p}(t) = \sum_{i=0}^{n} \mathbf{p}_i B_i^n(t)$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$



1.2.5 Bezier Surfaces

Definition 1.5: Bezier Surfaces

A **Bezier Surface** is an extension of Bezier Curves to the 3D space. Similarly it is a weighted combination of basis functions. The weights $\mathbf{p}_{i,j}$ are now 3 dimensional vectors and are the control points that define the control polygon.

Instead of having just t varying and having $\mathbf{p}$ be a function of one variable we now have u and v and $\mathbf{p} : \mathbb{R}^2 \to \mathbb{R}^3$.

(B_i^n is same definition as above).

$$\mathbf{p}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{p}_{i,j} B_i^m(u) B_j^n(v)$$

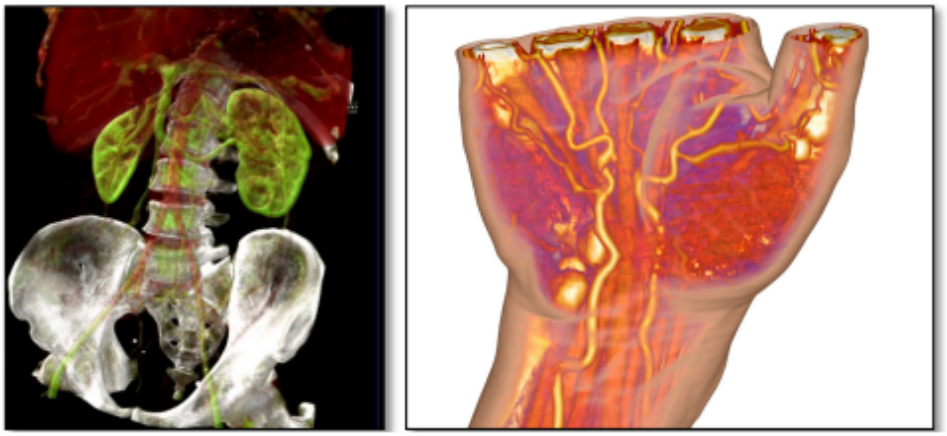
1.2.6 Volumetric Representations

Definition 1.6: Volumetric Representations

Volumetric Representations of geometry can be thought of as fields scalar fields in $\mathbb{R}^3$ where each point has a value that determines a property that we are interested in for example in MRI's it may be the density of tissue at that point. It may also be a colour.

It still follows the general form of parametric curves and surfaces with $m = 3$ and $n = 1$, but rather than mapping from a lower dimension to a higher dimension it is in reverse mapping from $\mathbb{R}^3$ to just a single real value representing the scalar quantity.

$$f : X \to Y, X \subseteq \mathbb{R}^3, Y \subseteq \mathbb{R}$$



1.2.7 Parametric Curve & Surfaces Pros and Cons

+ Easy to generate points on a curve/surface
+ Easy point-wise differential properties
+ Easy to control by hand
− Hard to determine inside/outside
− Hard to determine if a point is on a curve/surface
− Hard to generate by reverse engineering

1.3 Meshes

Definition 1.7: Meshes

Meshes in CG are points in 2D or 3D connected by edges forming polygons, e.g. triangles. They represent the boundary of an object as a discrete surface.

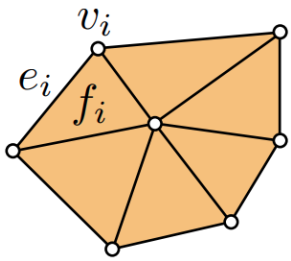
A polygonal mesh is a piecewise linear approximation of an underlying continuous surface. As the number of polygons increases the mesh approaches the true surface. Very quickly which is very useful.
A surface mesh is a graph embedded in 3D to go from a graph to a mesh we need postitions $\mathbf{p}$ for each vertex.

Each vertex in V is associated with a point p in 3D.

The vertices are connected by edges in E.

The edges form faces stored in a set F.

$$
\begin{aligned}
V &= \{v_1, \cdots, v_n\} \\
E &= \{e_1, \cdots, e_k\}, \qquad e_i \in V \times V \\
F &= \{f_1, \cdots, f_m\}, \qquad f_i \in V \times V \times V \\
\\
P &= \{\mathbf{p}_1, \cdots, \mathbf{p}_n\}, \qquad \mathbf{p}_i \in \mathbb{R}^3
\end{aligned}
$$



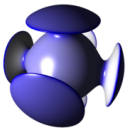
1.4 Implicit curves & Surfaces

Definition 1.8: Implicit Objects

$$f : \mathbb{R}^m \to \mathbb{R}$$

Implicit objects are the set of zeroes of a function, $m = 2$ for a curve in 2D or $m = 3$ for a surface or curve in 3D

Planar Curves 2D	Surfaces in 3D
$S = \{\mathbf{x} \in \mathbb{R}^2 \mid f(\mathbf{x}) = 0\}$	$S = \{\mathbf{x} \in \mathbb{R}^3 \mid f(\mathbf{x}) = 0\}$



To define a solid object the implicit function can be extended to include the inside of the surface.

$\{x \in \mathbb{R}^m \mid f(x) > 0\}$	Outside
$\{x \in \mathbb{R}^m \mid f(x) = 0\}$	The Surface/Curve
$\{x \in \mathbb{R}^m \mid f(x) < 0\}$	Inside

To shift a function to a new center that is not the origin it is as simple as:

$$f(\mathbf{x}) \text{ Shifted from origin to } \mathbf{x}_0 \text{ is } f(\mathbf{x} - \mathbf{x}_0)$$

1.4.1 Implicit Examples

For example for a circle centered at origin the implicit equation is $f(x, y) = x^2 + y^2 - r^2$ and for a sphere at the origin it is $f(x, y, z) = x^2 + y^2 + z^2 - r^2$

The surface normal is calculated by taking the derivatives of f.

$$\mathbf{n} = \nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)^T$$

The normal $\mathbf{n}$ obtained is not normalised so to normalise

$$\hat{\mathbf{n}} = \frac{\nabla f(x, y, z)}{||\nabla f(x, y, z)||}$$

For the case of a sphere:

$$\begin{aligned} \hat{\mathbf{n}} = \frac{\nabla f(x, y, z)}{||\nabla f(x, y, z)||} &= \frac{(2x, 2y, 2z)^T}{\sqrt{4x^2 + 4y^2 + 4z^2}} \\ &= \frac{(x, y, z)^T}{\sqrt{x^2 + y^2 + z^2}} \end{aligned}$$

So for a sphere with radius 1 the surface normal is just the position vector.

1.4.2 Implicit Object Pros and Cons

\+ Easy to determine inside/outside
\+ Easy to determine if a point is on a curve/surface
\+ Easy to combine
− Hard to generate points on a curve/surface
− Limited set of surfaces
− Does not lend itself to real-time rendering

1.5 Point Set Surfaces

One practical representation that combines some advantages of parametric and implicit representations are point set surfaces.
A point set surface is defined in terms of a point cloud in 3D.
A point cloud is a set of points possibly with a point-wise attribute (position, surface normal, colour etc.).
The data acquired normally comes from real-world, so quite important in practice.
One particular form a point set surface is defined via locally fitting a proxy surface

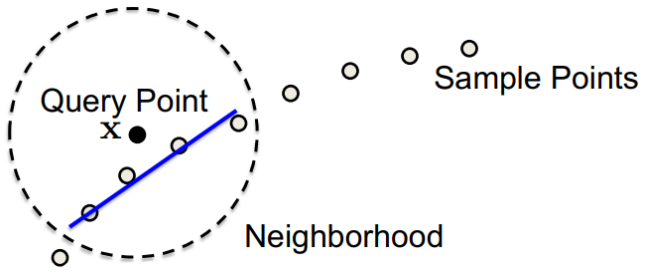
To go from a point set to an implicit function. One way is **Local Fitting**.

1.5.1 Local Fitting

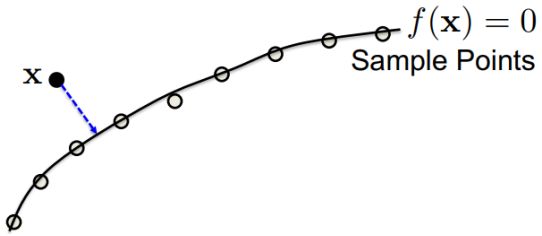
Definition 1.9: Local Fitting

Locally just looking at any point in space the surface patch of the point set closest to the point can be estimated as a simple object, in 3D a plane.

In the case of the curve a simple line.



The distance to the surface can then be approximated with the distance to this local proxy surface. Which is a lot faster than trying to use the point set explicitly.



The local approximations combine to make a global approximation, it has **fast projection** of points onto the surface. This is done by looking at the point and its nearest neighbours and project onto the local plane or line. This means, as opposed to implicit functions, its easier to generate points on the surface (or approximations of points on the surface)

1.5.2 Point Set Surfaces Pros and Cons

\+ Easy to determine inside/outside
\+ Easy to determine if a point is on a curve/surface
\+ Easy to generate points on the curve/surface
\+ Suitable for reconstruction from general data
\+ Direct real-time rendering (although not as fast as triangle meshes)
\+ Easily converted to a triangle mesh
\+ Robust to noise
− Not efficient to use in some modelling tasks

2 Discrete Differential Geometry

2.1 Manifolds

Definition 2.1: Open-Surfaces

An **Open Ball** of radius r and center $\mathbf{x}$ is the set of all points $\mathbf{p}$ such that they are a distance **less than** r away from $\mathbf{x}$

$$|\mathbf{x} - \mathbf{p}| < r$$

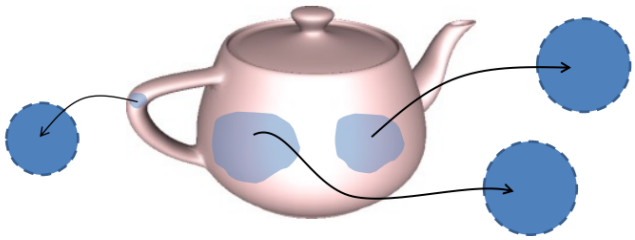
Similarly an **Open Disk** of radius R and center $\mathbf{x}$ is the set of all points on the plane at a distance **less than** R from $\mathbf{x}$.

Definition 2.2: Manifolds

A **Surface** is a **Closed 2-Manifold** if it is locally homeomorphic to a disk everywhere.

(More formally)
For every point $\mathbf{x}$ in M, there is an **Open Ball** $B_x(r)$ of radius r centred at $\mathbf{x}$ such that $M \cap B_x(r)$ is homeomorphic to an open disk.

(Less formally)
Locally everywhere on the surface is a plane.



2.1.1 Manifolds with Boundaries

Definition 2.3: Manifolds With Boundaries

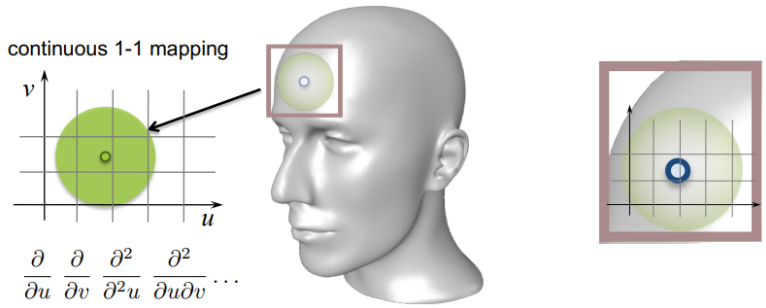
If all points at boundaries are homeomorphic to a half-disk (semi-circle), and points not at boundaries obey the 2-manifold principle of being homeomorphic to an open disk locally then that surface is a manifold with boundaries.

Having these properties allows for differential geometry.

2.2 Differential Geometry Properties

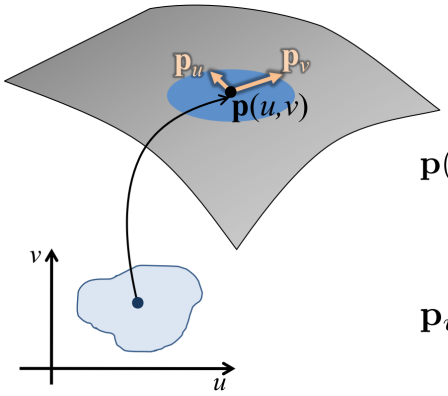
For every point on the surface you can define a local parameter space, which is local parametric representation of the surface. And this always works for manifolds. Similar to Parametric Surfaces but just locally.

Having the local parametric representation allows us to do partial derivatives with respect to u and v to obtain normals and other useful stuff.



2.2.1 Local Coordinates

To define local coordinates we have a surface patch and locally its a disk, and from the manifold properties we can define a u-v domain and from the parametric surfaces before we can define a function $\mathbf{p}(u, v)$ mapping from the u-v to points on the surface.



$$\mathbf{p}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}, \quad (u,v) \in \mathbb{R}^2$$

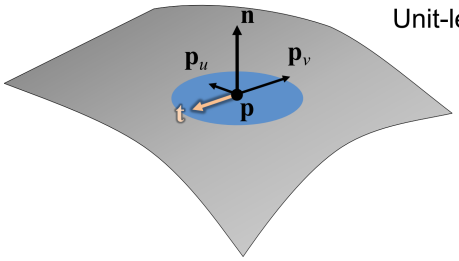
$$\mathbf{p}_u = \frac{\partial \mathbf{p}(u,v)}{\partial u}, \quad \mathbf{p}_v = \frac{\partial \mathbf{p}(u,v)}{\partial v}$$

Like seen in Parametric Surfaces $\mathbf{n}$ can be obtained with these two vectors $\mathbf{p}_u$ and $\mathbf{p}_v$

$$\mathbf{p}_u \times \mathbf{p}_v \neq \mathbf{0}$$

$$\hat{\mathbf{n}}(u,v) = \frac{\mathbf{p}_u \times \mathbf{p}_v}{||\mathbf{p}_u \times \mathbf{p}_v||}$$

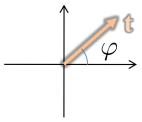
2.2.2 Normal Curvature



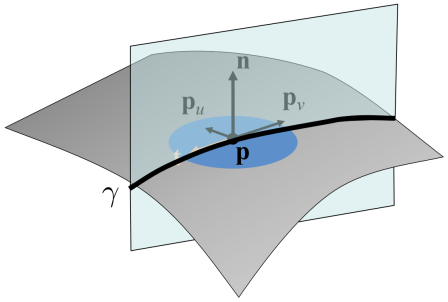
Unit-length $\mathbf{t}$ in the tangent plane

If $\mathbf{p}_u$ and $\mathbf{p}_v$ are orthogonal:

$$\mathbf{t} = \cos\varphi \frac{\mathbf{p}_u}{\|\mathbf{p}_u\|} + \sin\varphi \frac{\mathbf{p}_v}{\|\mathbf{p}_v\|}$$



$\mathbf{p}_u$ and $\mathbf{p}_v$ are basis vectors so any point on the local disk can be represented as a linear combination of the two.
As φ varies the $\mathbf{t}$ rotates around $\mathbf{n}$ in the tangent plane. For each of these rotated $\mathbf{t}$'s there is a different curvature.



The curve γ is the intersection of the surface with the plane through $\mathbf{n}$ and $\mathbf{t}$.

Normal curvature:

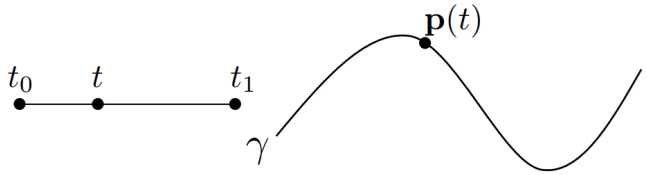
$$\kappa_n(\varphi) = \kappa_n(\gamma(\mathbf{p}))$$

2.2.3 Planar Curves (New stuff this year)

Looking at the the intersection we get a planar curve for each direction.

Like seen previously they can be paramterised by a single paramter t.

$$\mathbf{p}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \; t \in [t_0, t_1]$$



A given curve can have infinitely many paramerisations.

Imagine walking along the curve the speed of the walk can change at any point from one end to the other.

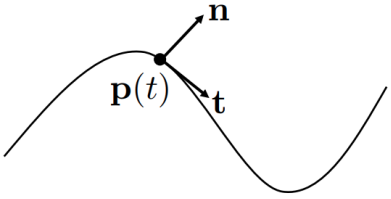
We choose a specific paramerisation, where the absolute value of the derivative w.r.t. t is a constant.

$$\|\frac{\partial \mathbf{p}(t)}{\partial t}\| = 1$$

With such a so-called arc-length parametrisation, we can define the tangent vector, normal vector, and curvature in terms of the first and second derivatives of the curve with simplified forms.

$$\mathbf{t} = \frac{\partial \mathbf{p}(t)}{\partial t}$$

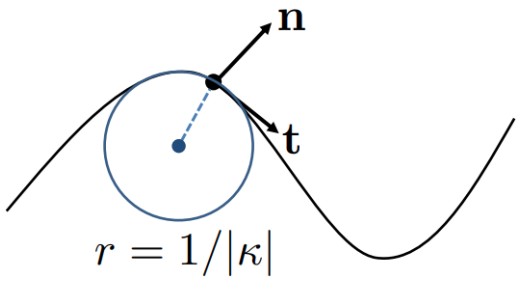
$$\frac{\partial^2 \mathbf{p}(t)}{\partial t^2} = \kappa(t)\mathbf{n}(t)$$



Definition 2.4: Osculating/Best-Fitting Circle

An **Osculating Circle** is a circle with radius equal to the inverse of the absolute value of the curvature at the point **p**.

Its essentially the circle that best approximates the region around **p**.



The best way to approximate this **Osculating** circle for a parametric curve is to have two points either side of **p**, then when this points converge to **p** then the circle defined by the 3 points is the best fittting circle at **p**.

$$\text{The circle defined by:} \quad \lim_{\epsilon \to 0^+} \mathbf{p}(t-\epsilon), \mathbf{p}(t), \mathbf{p}(t+\epsilon)$$

`https://www.desmos.com/calculator/qd0bayp351`
Desmos playground for curvature, osculating circles, tangents and normals.

Definition 2.5: Principal Curvatures

The two **Principal Curvatures** are the minimum and maximum curvatures at a point on a surface. The minimum and maximum are defined below.

Definition 2.6: Surface Curvatures

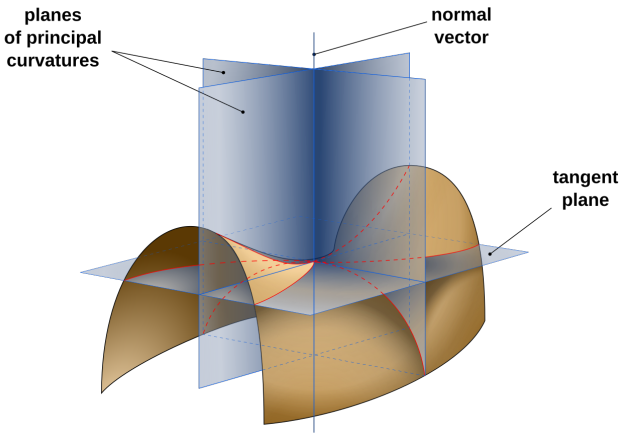
Minimal Curvature	$\kappa_1 = \kappa_{\min}$	$= \min_{\varphi} \kappa_n(\varphi)$
Maximal Curvature	$\kappa_2 = \kappa_{\max}$	$= \max_{\varphi} \kappa_n(\varphi)$
Mean Curvature	$H = \frac{\kappa_1 + \kappa_2}{2}$	$= \frac{1}{2\pi} \int_0^{2\pi} \kappa_n(\varphi)\, \mathrm{d}\varphi$
Gaussian Curvature	$K = \kappa_1 \cdot \kappa_2$	

Theorem 2.1: Euler's Theorem

Planes of principal curvature are **orthogonal** and independent of parameterization.

$$\kappa_n(\varphi) = \kappa_1 \cos^2 \varphi + \kappa_2 \sin^2 \varphi$$
$$\varphi = \text{angle with } \mathbf{t}_1$$

Euler's Theorem says that we can write any curvature as a linear combination of the principal curvatures. φ is the angle between the direction for the curvature and that for minimum normal curvature.



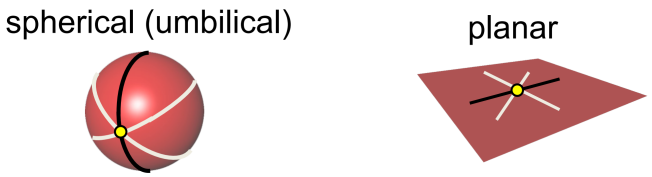
Proof 2.1: Mean Curvature

$$\begin{aligned}
H &= \frac{1}{2\pi}\int_0^{2\pi} \kappa_n(\varphi)\,\mathrm{d}\varphi \\
&= \frac{1}{2\pi}\int_0^{2\pi} \kappa_1 \cos^2(\varphi) + \kappa_2 \sin^2(\varphi)\,\mathrm{d}\varphi \\
&= \frac{1}{2\pi}\int_0^{2\pi} \kappa_1(1-\sin^2(\varphi)) + \kappa_2 \sin^2(\varphi)\,\mathrm{d}\varphi \\
&= \frac{1}{2\pi}\int_0^{2\pi} \kappa_1 + (\kappa_2-\kappa_1)\sin^2(\varphi)\,\mathrm{d}\varphi \\
&= \kappa_1 + \frac{1}{2\pi}\int_0^{2\pi} (\kappa_2-\kappa_1)\sin^2(\varphi)\,\mathrm{d}\varphi \\
&= \kappa_1 + \frac{1}{2\pi}\int_0^{2\pi} \frac{1}{2}(\kappa_2-\kappa_1)(1-\cos(2\varphi))\,\mathrm{d}\varphi \\
&= \kappa_1 + \frac{1}{2}(\kappa_2-\kappa_1) + \frac{1}{2\pi}\int_0^{2\pi} -\cos(2\varphi)\,\mathrm{d}\varphi \\
&= \frac{\kappa_1+\kappa_2}{2}
\end{aligned}$$

Definition 2.7: Isotropic Shapes

Isotropic means all directions are principal directions.

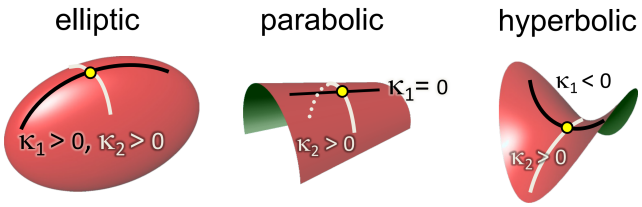
Shapes with an Gaussian Curvature $K > 0$, $\kappa_1 = \kappa_2$ are spherical (umbilical). If $K = 0$ then the surface is planar.



Definition 2.8: Anisotropic

Anisotropic means that there are 2 distance principal directions at any point on the surface

If $K > 0$ then the shape is elliptic
If $K = 0$ then the shape is parabolic
If $K < 0$ then the shape is hyperbolic



2.2.4 Laplace-Beltrami

Definition 2.9: Laplace Operator

Also known as the Laplacian.

$$f\colon \mathbb{R}^3 \to \mathbb{R} \qquad \Delta f\colon \mathbb{R}^3 \to \mathbb{R}$$

$$\Delta f = \operatorname{div} \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$$

This sum of second order derivatives is due to the following:

$$\operatorname{grad} f = \nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

$$\operatorname{div} \mathbf{F} = \nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$$

This Laplace operator can be extended to manifold surfaces. Lets assume there's a function that lives on the surface, which is not a Euclidean space in general. We then have the same operators defined slightly differently. This is a generalization of the Laplace operator and is called the Laplace-Beltrami operator.

Definition 2.10: Laplace-Beltrami Operator

$$f\colon \mathcal{M} \to \mathbb{R}$$
$$\Delta f\colon \mathcal{M} \to \mathbb{R}$$

$$\Delta_{\mathcal{M}} f = \operatorname{div}_{\mathcal{M}} \nabla_{\mathcal{M}} f$$

$\Delta_{\mathcal{M}}$ is the **Laplace-Beltrami operator**. f is a function on surface $\mathcal{M}$.

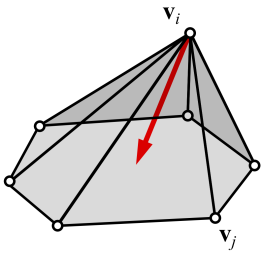
An interesting property of the operator is if you apply it to the coordinate function (i.e. $f(x, y, z) = x$) of a point on the surface. You get the scaled surface normal at that point. The scaling is exactly -2 times the mean curvature.

$$\Delta_{\mathcal{M}} \mathbf{p} = \operatorname{div}_{\mathcal{M}} \nabla_{\mathcal{M}} \mathbf{p} = -2H\hat{\mathbf{n}} \in \mathbb{R}^3$$
$$\Delta_{\mathcal{M}} \mathbf{p} = -2H\hat{\mathbf{n}}$$

This operator is so important due this property as it allows us to define curvature and normal in terms of the the application of the operator to the coordinate function. If we can define the application to the coordinate functions on a discrete surface, we can get the mean curvature and surface normal

2.3 Discrete Laplace-Beltrami

$$\Delta_{\mathcal{M}}\mathbf{p} = -2H\hat{\mathbf{n}}$$



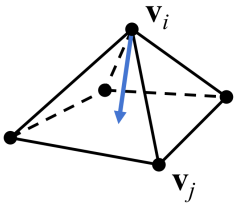
We want to take the Laplace-Beltrami at the point $\mathbf{v}_i$, to do this we take the average of its connected neighbours location to get an approximation of the result and in turn an approximation of the mean curvature and surface normal.

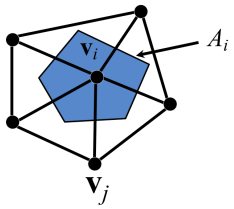
$$\begin{aligned} L_u(\mathbf{v}_i) &= \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} (\mathbf{v}_j - \mathbf{v}_i) \\ &= \left(\frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j \right) - \mathbf{v}_i \end{aligned}$$

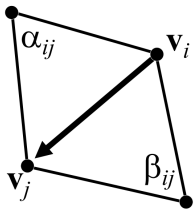
The direction will be an approximation of the negative normal $-\hat{\mathbf{n}}$ and the length will be $\approx 2H$

To get a closer approximation of the mean curvature and surface normal the directions can be weighted. One general scheme for triangular meshes is using cotangent weights.

$$L_c(\mathbf{v}_i) = \frac{1}{A_i} \sum_{j \in \mathcal{N}(i)} \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij})(\mathbf{v}_j - \mathbf{v}_i)$$

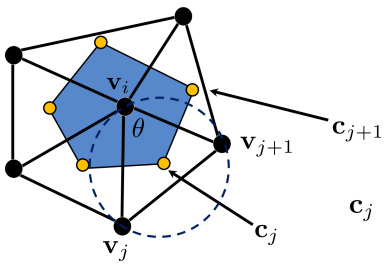






$$\mathbf{c}_j = \begin{cases} \text{Circumcentre of } \triangle(\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_{j+1}) & \text{if } \theta < \frac{\pi}{2} \\ \text{Midpoint of edge } (\mathbf{v}_j, \mathbf{v}_{j+1}) & \text{if } \theta \geq \frac{\pi}{2} \end{cases}$$

$$A_i = \sum_j \text{Area } (\triangle(\mathbf{v}_i, \mathbf{c}_j, \mathbf{c}_{j+1}))$$



2.3.1 Discrete Laplace-Beltrami Relation to Normal & Curvature

Definition 2.11: Discrete Laplace-Beltrami Formulas

Mean Curvature (Sign according to normal)

$$|H(\mathbf{v}_i)| = ||L_c(\mathbf{v}_i)||/2$$

Gaussian Curvature

$$K(\mathbf{v}_i) = \frac{1}{A_i}\left(2\pi - \sum_j \theta_j\right)$$

Principal Curvatures

$$\kappa_1 = H - \sqrt{H^2 - K} \qquad \kappa_2 = H + \sqrt{H^2 - K}$$

Proof 2.1: Principal Curvature Formulas

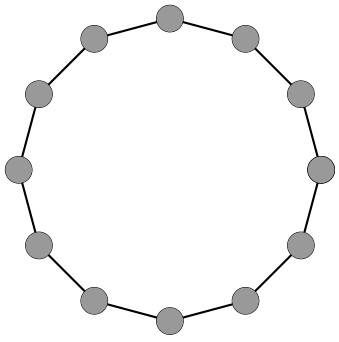
Using the facts $H = \frac{\kappa_1 + \kappa_2}{2}$ and $K = \kappa_1\kappa_2 \implies \kappa_2 = \frac{K}{\kappa_1}$

$$\begin{aligned}
& 2H = \kappa_1 + \kappa_2 \\
\implies & 2H = \kappa_1 + \frac{K}{k_1} \\
\implies & 2H\kappa_1 = \kappa_1^2 + K \\
\implies & (k_1 - H)^2 = H^2 - K \\
\implies & \kappa_1 = H \pm \sqrt{H^2 - K} \qquad \text{Similarly } \kappa_2 = H \pm \sqrt{H^2 - K}
\end{aligned}$$

Since $\kappa_1 \leq \kappa_2$

$$\kappa_1 = H - \sqrt{H^2 - K} \qquad \kappa_2 = H + \sqrt{H^2 - K} \qquad \square$$

By considering meshes as a discrete surface we could get an approximation of the Laplace-Beltrami operator. By doing the same for graphs or point clouds we can define an approximation of the application of the Laplace-Beltrami operator to the coordinate function and approximate the curvatures.
In this case, the approximation is with Gaussian weights instead of Cotangent weights.



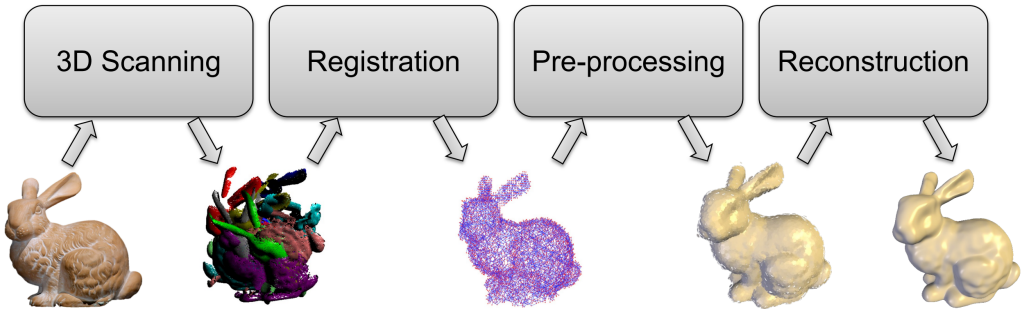
$$\begin{aligned}
h_t(x_i, x_j) &= e^{-d(x_i,x_j)/t} \\
&= e^{-||\mathbf{x}_i - \mathbf{x}_j||^2/t}
\end{aligned}$$

$$L_g(x_i) = \frac{1}{\sum_{j=1}^n h_t(x_i, x_j)} \sum_{j=1}^n h_t(x_i, x_j)(x_j - x_i)$$

This approximation allows us to capture complex domains, in fact any domain that can be point sampled.

3 Geometry Processing

3.1 Shape Acquisition



It starts with **3D scanning** with a depth camera or other camera. Going from a real object to a 3D scan and each scan is a 3D point cloud (shown in different colours)

Then **Registration** means aligning these point clouds into a common coordinate frame so they make a coherent point cloud of the object.

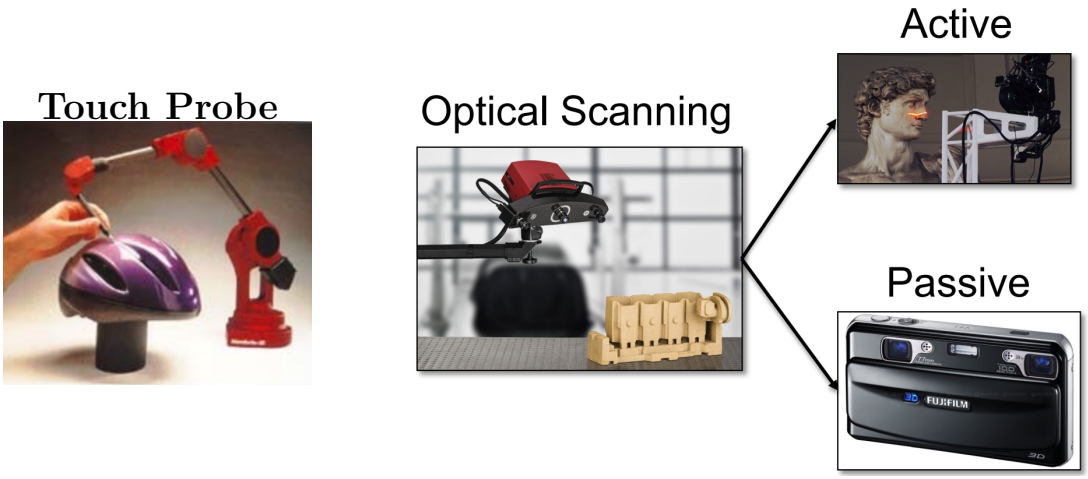
Then **Pre-Processing** is cleaning up the point cloud so it is nicer to work with later. This could be reducing the number of points or removing noise from the scans.

Then **Reconstruction** from the point cloud to create a mathematical representation of the object which can then be formed into a mesh.

3.1.1 3D Scanning

Touch Probes can be used to capture 3D objects, they are **precise** in capturing the large details of the object However they **cannot accurately capture small objects** or fine detail on large objects, and shapes with cavities. Soft squishy objects such as faces cannot be captured using this method as they touch probe would deform the surface.

Optical Scanning is **very fast** as it is essentially just taking multiple pictures of the object to capture however the scanning may be affected by the finish on the surface for example if its **glossy**. To help with this problem paint can be sprayed onto the surface so that it has more diffuse lighting as opposed to the spectral reflections that cause the problems.



There's two main types of Optical Scanning, those are **Active** and **Passive** scanning.

Definition 3.1: Active Scanning

Active Scanning is where the scanner has a light source that is controllable such as a laser or IR.

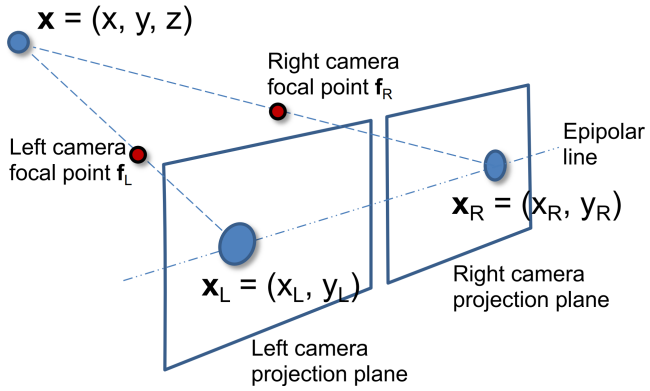
Examples of Active systems

LIDAR - Measures the time it takes for a laser beam to hit the object and come back to the sensor.

Triangulation Laser - Projected laser beam is photographed, giving the distance of the pattern. For example a line may be projected horizontally onto the subject and the deformation of the line gives the distance. (Shown in the active optical scanning image above)

Definition 3.2: Passive Scanning

Passive Scanning is where the system only has a scanner and there is no control over the environment and the object poses.



The idea is you have multiple cameras typically RGB colour cameras, at least two. And then the rotation of each camera with respect to each other. Each point $\mathbf{x}$ maps to two different points, $\mathbf{x}_\mathrm{L}$ and $\mathbf{x}_\mathrm{R}$. We can then find $\mathbf{x}$ by intersecting the lines connecting $\mathbf{f}_\mathrm{L}$ and $\mathbf{x}_\mathrm{L}$, and $\mathbf{f}_\mathrm{R}$ and $\mathbf{x}_\mathrm{R}$.

This assumes we already know $\mathbf{x}_\mathrm{L}$ and $\mathbf{x}_\mathrm{R}$ corresponds to the same point in 3D.

3.1.2 Registration

Definition 3.3: Registration

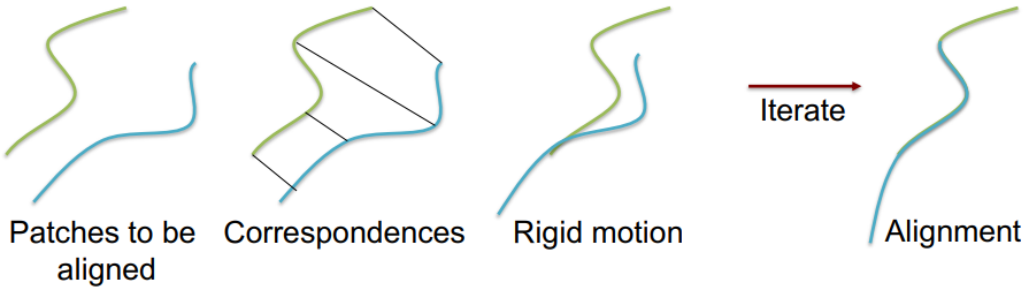
Registration is when multiple scans of a single object which yield individual point clouds are brought into a common coordinate frame to create a single point cloud. There are a few methods such as **Iterative Closest Point Algorithms** or **Feature-Based Methods**.

This is because the individual point clouds are in different coordinate frames for example rotated and translated.

Iterative Closest Point Algorithms - (ICP) Algorithms

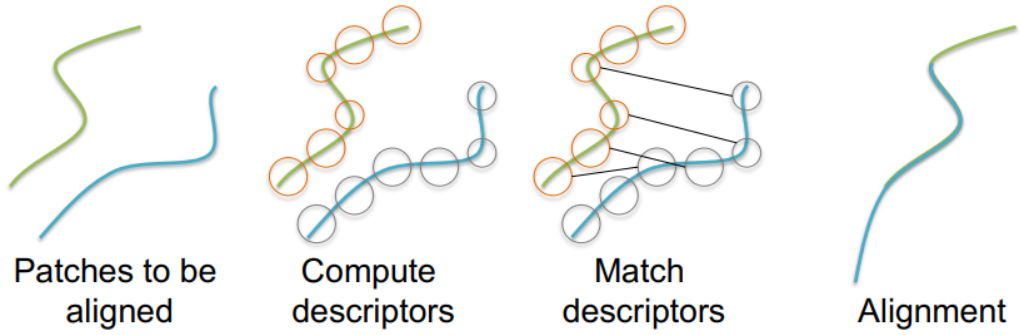
ICP algorithms are typically used for precise registration. The steps are below:

1. Establish point-to-point correspondences
2. Compute and apply the optimum rotation and translation
3. Iterate the previous steps



Feature-Based Methods

If the rotation and or translation of the patches are initially really off, then establishing the correspondences for ICP algorithms is not easy and most likely will fail to align the patches.
Feature-Based Methods compute rotation and translation invariant descriptors for points and match those.



Normally we use curvature of the surface or the curve in this example for these features.

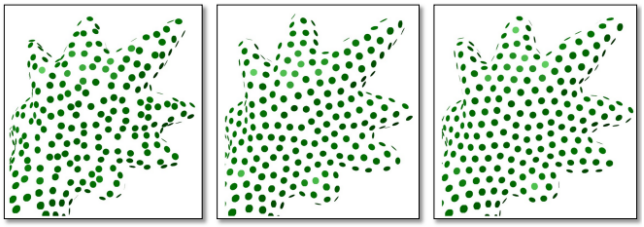
3.1.3 Pre-Processing

Definition 3.4: Pre-Processing

Pre-Processing is the act of cleaning, repairing, resampling of the point clouds.

Accurate and efficient reconstructions of surfaces fundamentally depends on how we distribute points on the surface to represent the surface.

There are extensions of the sampling theorem to irregular sampling on surfaces.



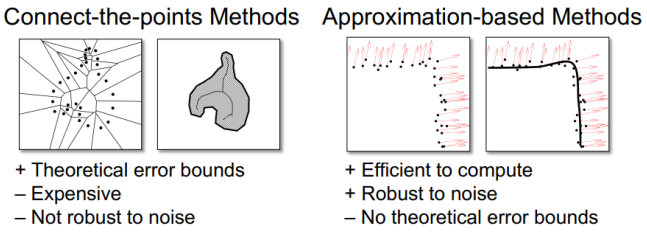
3.1.4 Reconstruction

Definition 3.5: Reconstruction

Reconstruction is turning the point cloud to a mathematical representation of the shape.

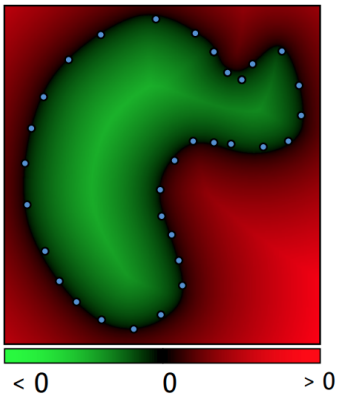
There are two ways to reconstruct a mathematical definition of a surface.

1. Connecting the sample points with lines/polygons
2. Approximate the underlying surface for example with a least squares solution



The connect the points creates a curve that passes through every point whereas the approximation method does not necessarily create a curve that passes through the points.

Implicit Approximation
Some of the most versatile reconstruction methods work with implicit representations



We try to estimate the points as an implicit function then extract the zero set

$$\{\mathbf{x}|f(\mathbf{x}) = 0\}$$

3.2 Texture Mapping

Once the surface has been reconstructed we can add colour and specific details using texture mapping.
To do this it requires parametrizing the surface, i.e. finding a mapping from a plane to the surface.

The (u, v) coordinates live on the image plane. For each (u, v), we have the corresponding point (x, y, z). We can assume the texture space is a unit square and define a map P from the surface to this square.

$$P : M \to [0,1] \times [0,1]$$
$$P(x, y, z) = (u, v)$$

Then there is a further mapping from the (u, v) space to an attribute, e.g. colour.
This mapping will be denoted by T.

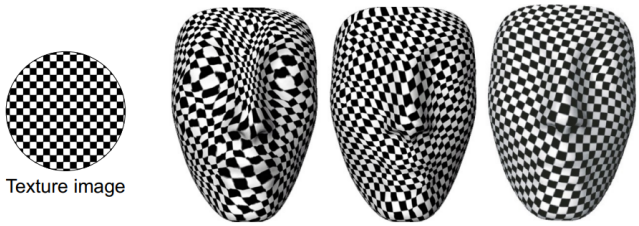
$$T : [0,1] \times [0,1] \to \text{RGB}$$
$$T(u, v) = (\text{r}, \text{g}, \text{b})$$

These two individual mappings can be combined to create a final mapping which provides us with a colour for each point on the surface.

$$\text{Colour}(x, y, z) = T(P(x, y, z))$$

3.2.1 Parametrization

The main part here is the parameterization P. In general, it is impossible to have a distortion-free map from a plane to a surface in 3D. Therefore we need to decide on what distortion we allow for.

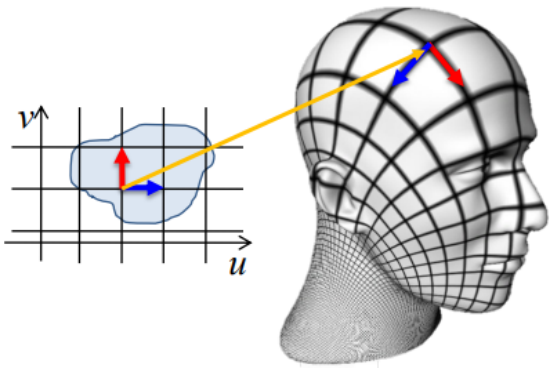


To define the properties of distortion, we first write $\mathbf{p}$ as three functions, one for each component as we did before. Such a parameterization implies a map from uv-space to those in 3D space.

$$\mathbf{p}(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}, \ (u, v) \in \mathbb{R}^2$$

We can then define the so-called **First Fundamental** form which is the 2×2 matrix consisting of dot products of derivatives with respect to u and v.

$$\mathbf{p}_u = \frac{\partial \mathbf{p}(u, v)}{\partial u}, \ \mathbf{p}_v = \frac{\partial \mathbf{p}(u, v)}{\partial v}$$
$$\mathbf{I} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} \mathbf{p}_u^T \mathbf{p}_u & \mathbf{p}_u^T \mathbf{p}_v \\ \mathbf{p}_u^T \mathbf{p}_v & \mathbf{p}_v^T \mathbf{p}_v \end{bmatrix}$$



(^{T} denotes the dot product, implying the transpose operator followed by matrix multiplication)

This matrix measures the angle and length change in different terms.
The area distortion can also be expressed in terms of these terms. It is defined as the change in areas when applying the map.

$$\text{Angle Change} = F = \mathbf{p}_u^T \mathbf{p}_v$$
$$\text{Length Change} = G = \mathbf{p}_v^T \mathbf{p}_v$$

and

Definition 3.6: Area Distortion

Area Distortion: $dA = \sqrt{EG - F^2}\, du\, dv$

Definition 3.7: Conformal Parametrization

Conformal maps preserve angles such that for example squares map to squares with right angles on the surface. For conformal maps, the **First Fundamental** form is the identity matrix.

In practice we can typically compute approximately conformal maps for parameterization.

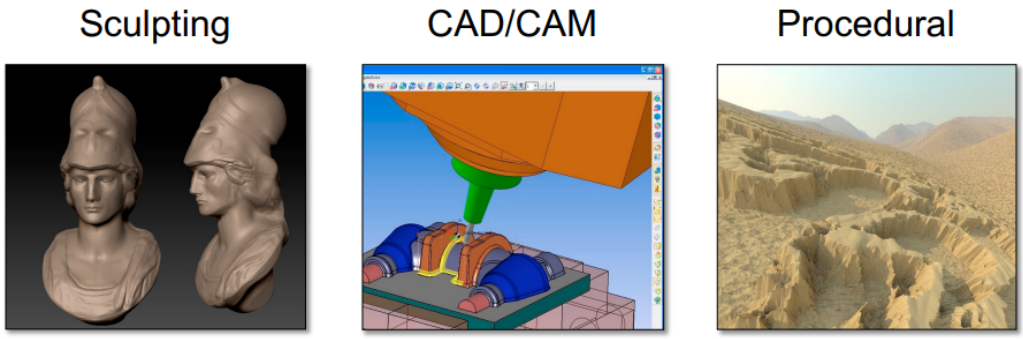
$$\mathbf{I} = \begin{bmatrix} \mathbf{p}_u^T \mathbf{p}_u & \mathbf{p}_u^T \mathbf{p}_v \\ \mathbf{p}_u^T \mathbf{p}_v & \mathbf{p}_v^T \mathbf{p}_v \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

3.3 Editing Geometry

3.3.1 Modelling Tools

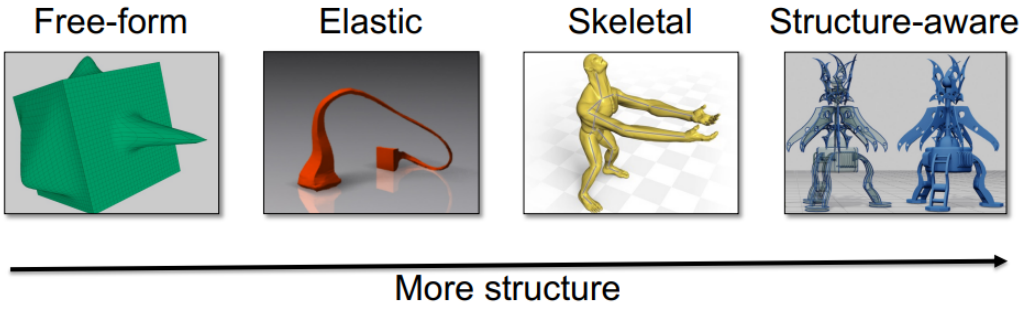
There are many established tools to further edit geometry they depend on how you want to edit it and for what reason.

1. Some are artistic for example via **Sculpting** where it mimics how you would do it in real life
2. Others are for engineering related environments for example designing machine parts. **CAD/CAM**
3. **Procedural** is where rules are defined for the geometry to be created. It is particularly useful in games for creating terrain or cities.



3.3.2 Deformations

Deforming an initial surface is a typical workflow in digital modelling tools.
Such deformations can be based on low-level related characteristics for example elastic or plastic deformations or high level deformations such as skeletal deformations.

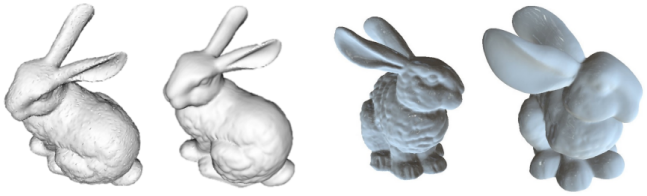


3.3.3 Cutting and Fracturing

Cutting and Fracturing through physics simulations for example leads to changing mesh connectivity.
This is rather difficult and one of the reasons why connectivity-free geometry representations are useful.

3.3.4 Smoothing and Filtering

Many algorithms exist for smoothing geometry or amplifying certain features of surfaces.
A lot of them are curvature-based, for example reducing local curvature across the mesh leads to smoothing.



3.3.5 Compression and Simplification

Compression of geometric representations can be reducing the number of vertices in a mesh or quantizing the vertex locations, it is important for modern large data processing with billions of triangles.

Many compression methods, like Smoothing and Filtering, also can be reduced to preserving local properties given by curvature.

4 Animation

Definition 4.1: Animation

Technically: **Animation** is the process of creating sequences of images

Artistically: **Animation** is the process of bringing images to life.

Things we animate:

1. Characters
2. Faces
3. Hair
4. Elastic Materials
5. Natural Phenomena
6. Cloths
7. Herds
8. Crowds
9. etc.

How we Animate

1. Key-Framing
2. Skeletal
3. Subspace/Parameters
4. Physically Based
5. Motion Capture
6. Video Based
7. Example Based
8. Procedural
9. etc.

Key-framing controls given by abstract skeletons or other parameters is how animations are created.
This provides control over deformations and timing, which are both critical for creating animations.
The final complex animations are obtained by blending simpler transformations together.

There are many ways to augment this process, but the fundamental ideas of controls and blending transformations do not change too much.

4.1 Considerations

Animating requires absolute **control**.
Efficiency and **Style** are also tightly coupled with control.
Controlling animations with computers typically means dragging points and curves around to define deformations.

4.2 Character Animation

Character animation will be used a working example.

It is the most important and generalizable way to animate objects and heavily used in industry. It requires blending transformations, for which we need to understand some fundamental maths.

Animation Pipeline

1. Story boarding
   - A board for setting up the story
   - Helps planning animations
2. Concept Design
   - Sketches for characters and environment
   - Main features of the characters
3. 3D Modelling
   - Moving to computers
   - Geometry and Textures
4. Rigging
   - Embedding animation controllers
   - Construct a skeleton and attach additional controls
   - Key-frame the controllers for animation
5. Blend Shapes Creation
   - Create facial expressions
   - Used to generate other expressions via blending
6. Animation
   - Set key-frames for controllers
   - Steer interpolation and timing with time controls
7. Post-Effects
   - Other animations (fluids, cloth, etc)
   - Lighting and shading
   - Rendering

4.3 Rigging

Definition 4.2: Rigging

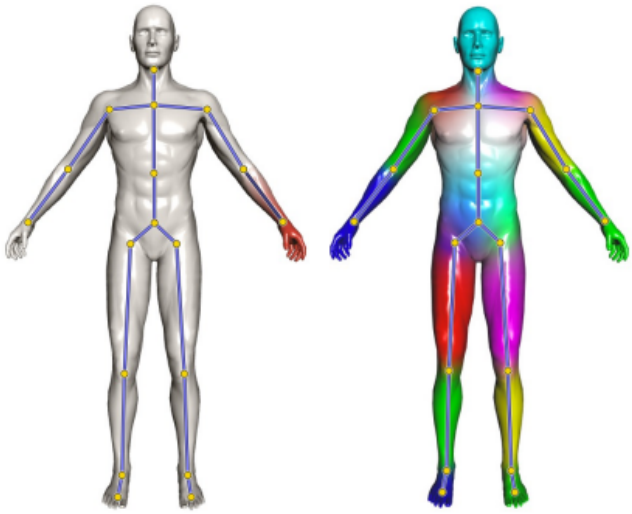
A 3D model is typically represented as a surface mesh. **Rigging** is the process of attaching an abstract skeleton consisting of bones and joints to the model.

As the user changes the configuration of the skeleton the model is deformed accordingly

Rigging in short is the process of attaching a skeleton to a model.

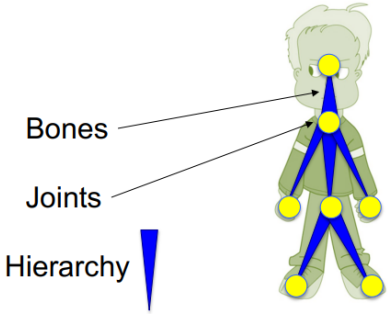
The first step to rigging is positioning the skeleton so that semantically corresponding parts of the skeleton and the model are close, i.e. the forearm bone and the forearm are roughly aligned.

The next step is defining a function on the surface model for each bone, which is colour coded below



If we look closer at the skeleton we have bones, joints, and the hierarchy of these in a skeleton. (This is just an abstract look at a skeleton for the purpose of user interpretation of motion for motion control)

The hierarchy is important for controlling the skeleton, e.g. if you pick the hip joint and change the location of it, the legs are moving along with it.



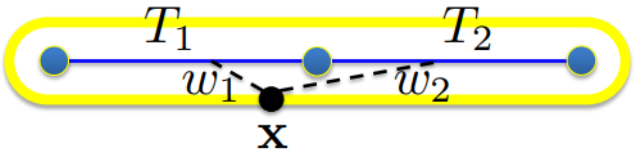
A rotation and a translation is stored on each bone or joint. We can assume they are stored on bones. As the user transforms bones via the mouse or another control these transformations are updated.

We can store the rotations as matrices $\mathbf{R}_i$ and vectors t_i as translations.

Note: This means each bone is rigid so no bending.

The second step of rigging is attaching each bone to the 3D model. This means computing a weighting function for each bone that determines how much the transformation on a bone is affecting a given point on the surface of the model

To see how this works we're going to look at a simple case: A simple mesh with two bones and 3 joints.



As we change the transformation $\mathbf{T}_i$, (The rotation and translation), the mesh should deform accordingly.

To compute a new position of a given point $\mathbf{x}$ on the mesh.

1. Blend the transformations with the weights stored w_i. (Denoted by the avg function below)
2. Apply the transformation to the point $\mathbf{x}$ to get the new corresponding point on the deformed model.

$$T(\mathbf{x}) = \text{avg}(T_1, T_2, w_1, w_2)$$

4.3.1 Linear Blend Skinning

One way to represent the transformations is via 4×4 transformation matrices $\mathbf{T}_i$, and blending via a linear combination of those.

Each matrix stores both the rotation and translation.

The final blended transformation is then applied to the point $\mathbf{x}$ in homogenous coordinates, i.e. $\mathbf{x} = [x, y, z, 1]^T$

$$\mathbf{T}(\mathbf{x}) = w_1(\mathbf{x})\mathbf{T}_1 + w_2(\mathbf{x})\mathbf{T}_2$$
$$\mathbf{x}' = \mathbf{T}(\mathbf{x})\mathbf{x}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}_{1,1} & \dots & \mathbf{R}_{1,3} & t_1 \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{R}_{3,1} & \dots & \mathbf{R}_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The main problem with linear blend skinning is that it assumes no structure for rotation matrices.
(In general, linear combinations of rotation matrices are not valid rotation matrices)

$$w_1\mathbf{T}_1 + w_2\mathbf{T}_2$$

- $w_1\mathbf{t}_1 + w_2\mathbf{t}_2$ is a valid translation
- $w_1\mathbf{R}_1 + w_2\mathbf{R}_2$ is **not** a valid rotation matrix

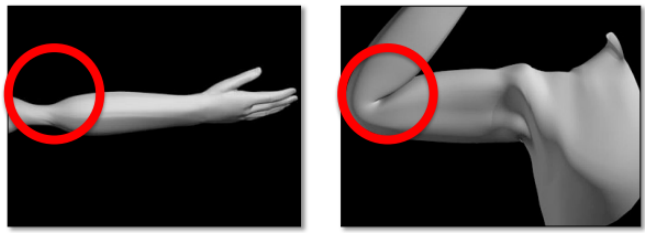
For a rotation matrix to be valid it should satisfy the following two properties

1. Its transpose is equal to its inverse $\mathbf{R}^T = \mathbf{R}^{-1}$
2. Its determinant is equal to 1 $\det \mathbf{R} = 1$

For linear blending though this is not the case.

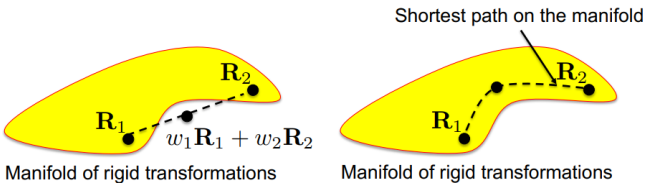
$$\begin{aligned} &(w_1\mathbf{R}_1 + w_2\mathbf{R}_2)^T \\ =&(w_1\mathbf{R}_1^T + w_2\mathbf{R}_2^T) \\ \neq&(w_1\mathbf{R}_1 + w_2\mathbf{R}_2)^{-1} \end{aligned}$$

In practice, invalid matrices lead to volume loss, especially around the joints where the blending is strongest.



Left: **Candy-Wrapper** artefact Right: **Elbow Collapse** artefact

We can visualize what is going wrong by considering the subspace (it is a manifold) of rotation matrices in the 9 dimensional space of all 3x3 matrices. A linear combination of rotation matrices will not necessarily lie on this manifold. Instead, what we want is the shortest path that respects the manifold of rotation matrices.



This manifold is called SO(3) and is given by the two condition on valid rotation matrices we have seen.

Manifold of Rotations - SO(3)

$$\mathbf{R}^T = \mathbf{R}^{-1}$$
$$\det \mathbf{R} = 1$$

Manifold of Rigid Transformations - SE(3)

$$\mathbf{R}^T = \mathbf{R}^{-1} \qquad \det \mathbf{R} = 1 \qquad \mathbf{T} = \begin{bmatrix} \mathbf{R} & & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In summary, it is hard to generate valid rigid transformations with linear combinations if we work with the matrix representation of transformations. Instead, we will utilize an alternative representation: **dual quaternions**.

4.4 Quaternions

Definition 4.3: Quaternion

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + \mathbf{s}\sin\left(\frac{\theta}{2}\right)$$

$\mathbf{q}$	is the quaternion
θ	is the **rotation angle**
$\mathbf{s}$	is the **rotation axis**

The rotation axis $\mathbf{s}$ is represented with an extension of imaginary numbers, denoted with i, j, k. The rotation axis is a unit vector.

$$\mathbf{s} = s_i i + s_j j + s_k k$$
$$s_i^2 + s_j^2 + s_k^2 = 1$$
$$i^2 = j^2 = k^2 = ijk = -1$$

4.4.1 Operations on Rotation Quaternions

(For quaternions that have a norm of 1)

Conjugate

The conjugate has the same meaning as for imaginary number i.e. the non-real part (in this case is $\mathbf{s}$) is negated.

$$\mathbf{q}^* = \cos\left(\frac{\theta}{2}\right) - \mathbf{s}\sin\left(\frac{\theta}{2}\right) = \cos\left(-\frac{\theta}{2}\right) + \mathbf{s}\sin\left(-\frac{\theta}{2}\right)$$

Inverse

$$\mathbf{q}^{-1} = \mathbf{q}^*$$

Multiplication

Composing rotations is done by multiplying quaternions. Multiplication is carried out by using the following rules

$$\begin{aligned} ij &= k & ji &= -k \\ jk &= i & kj &= -i \\ ki &= j & ik &= -j \end{aligned}$$

$$i^2 = j^2 = k^2 = -1$$

$$\mathbf{q}_1\mathbf{q}_2 = (a_1 + b_1 i + c_1 j + d_1 k)(a_2 + b_2 i + c_2 j + d_2 k)$$

Multiplying a quaternion with its conjugate gives us the norm, which is 1 for the case of quaternions representing rotations.

Norm

$$\|q\|^2 = \mathbf{q}\mathbf{q}^* = \cos^2\left(\frac{\theta}{2}\right) + \|\mathbf{s}\|^2 \sin\left(\frac{\theta}{2}\right) = 1$$

Power

$$\mathbf{q}^t = e^{t\log\mathbf{q}}$$

$$\log\mathbf{q} = \frac{\theta}{2}\mathbf{s} \qquad e^{\mathbf{q}} = \cos\|\mathbf{q}\| + \frac{\mathbf{q}}{\|\mathbf{q}\|}\sin\|\mathbf{q}\|$$

($e^{\mathbf{q}}$ only holds when the scalar part of the quaternion is 0)

Applying to Location Vectors

To apply a quaternion to a vector to rotate it, we first need to write the vector in i, j, k space.
The $\mathbf{v}'$ is then the rotated version of $\mathbf{v}$

$$\begin{aligned}\mathbf{v} &= v_i i + v_j j + v_k k \\ \mathbf{v}' &= \mathbf{q}\mathbf{v}\mathbf{q}^*\end{aligned}$$

Blending Quaternions

For the special case that the axes of rotation are the same for both quaternions, their blending/interpolation is the interpolation of the angle of rotation.

$$\mathbf{s} = \mathbf{s}_1 = \mathbf{s}_2$$

$$\text{interpolate}(\mathbf{q}_1, \mathbf{q}_2, t)$$

$$\begin{aligned}\theta(t) &= (1-t)\theta_1 + t\theta_2 \\ \mathbf{q}(t) &= \cos\left(\frac{\theta(t)}{2}\right) + \mathbf{s}\sin\left(\frac{\theta(t)}{2}\right)\end{aligned}$$

Spherical Blending:

For the general case though we use the so-called spherical blending for two quaternions.

For $t = 0$ we get $\mathbf{q}_1$, and for $t = 1$ we get $\mathbf{q}_2$ as expected.

$$\mathbf{s}_1 \neq \mathbf{s}_2$$

$$(\mathbf{q}_2\mathbf{q}_1^*)^t\mathbf{q}_1$$

Blending More Than Two Quaternions:
There is no closed form solution. A good approximation is a weighted sum of quaternions.
(This is the reason why we care about quaternions. In contrast to matrix based representations quaternion linear blending does not lead to volume loss and other artefacts)

$$\mathbf{q}_1, \cdots, \mathbf{q}_n \qquad w_1, \cdots, w_n$$

$$\mathbf{b} = \sum_{i=1}^{n} w_i \mathbf{q}_i$$

4.4.2 Dual Quaternions

Definition 4.4: Dual Numbers

Dual numbers have a real part and a **dual** part. The dual number ϵ has the property

$$\epsilon^2 = 0 \qquad \wedge \qquad \epsilon \neq 0$$

$$\hat{x} = x_0 + \epsilon x_\epsilon$$

This property leads to cancellation of higher order terms in multiplication. For an example multiplication of two dual numbers.

$$\begin{aligned} & (a_0 + \epsilon a_\epsilon)(b_0 + \epsilon b_\epsilon) \\ = \; & a_0 b_0 + \epsilon(a_0 b_\epsilon + a_\epsilon b_0) \end{aligned}$$

Definition 4.5: Dual Quaternions

For dual quaternions the component of the axis $\mathbf{s}$ become dual numbers.

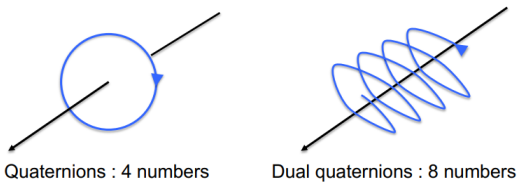
All operations and expressions we are interested in stay the same.

$$\hat{q} = \cos\left(\frac{\hat{\theta}}{2}\right) + \hat{\mathbf{s}}\sin\left(\frac{\hat{\theta}}{2}\right)$$

In particular, we still have the linear combination as a good approximation of proper shortest path blending.

$$\hat{\mathbf{b}} = \sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i$$

These dual quaternions now mean we can represent a rotation and a translation with a rotation around the axis $\mathbf{s}$ and a translation along the same axis.



There is one thing that was brushed over: only normalized quaternions/ dual quaternions represent valid rotations/ rigid transformations.

With normalization, we always get dual quaternions representing valid rigid transformations.

$$\hat{\mathbf{b}} = \sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i$$

Only generates valid transformations iff its normalized.

$$\hat{\mathbf{b}} = \frac{\sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i}{\left\| \sum_{i=1}^{n} w_i \hat{\mathbf{q}}_i \right\|}$$

An important property of the approximate blending is that it is invariant to coordinate change: If we first transform all dual quaternions with the same transformations and then blend the representing quaternions we get the same result as first blending them and then applying the transformation.

This follows from the property: $\Sigma w_i \hat{\mathbf{q}} \hat{\mathbf{q}}_i = \hat{\mathbf{q}} \Sigma \hat{\mathbf{q}}_i$

This simple linear blending stays very close to the shortest path blending on SE(3), the manifold of valid rigid transformations.

4.4.3 Challenges of Transformations

Apart from the transformation representation, there are challenges with how we compute the weight for blending. There are several intuitive properties.
Blending Transformations - We use Dual Quaternions

Weights $w_i(\mathbf{x})$

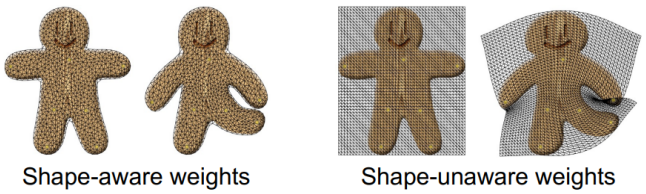
- Shape Adaptive
- Intuitive Deformations
- Smooth Deformations

The first property is partition of unity. This is important for volume preservation.

$$\sum_{i=1}^{n} w_i(\mathbf{x}) = 1$$

The second property is smoothness of weights. This ensures we get smooth deformations without artefacts.

Finally the last property is shape-awareness which leads to local influence with respect to respect to the deformed shape and is important for intuitive control.



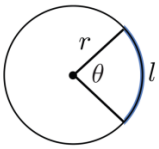
5 The Rendering Equation

5.1 Basic Definitions

Angle:

$$\theta = \frac{l}{r}$$

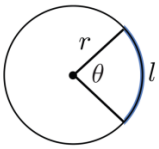
A circle is 2π radians



Solid Angle:

$$\Omega = \frac{A}{r^2}$$

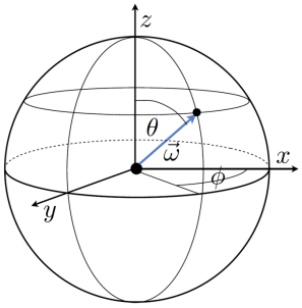
A sphere is 4π steradians



Direction

A point on the unit sphere parameterized by two angles θ as zenith and ϕ as azimuth.

$$\begin{aligned}\vec{w} &= (\theta, \phi)\\ \vec{w}_x &= \sin\theta\cos\phi\\ \vec{w}_y &= \sin\theta\sin\phi\\ \vec{w}_z &= \cos\theta\end{aligned}$$



$$\text{latitude} = \frac{90}{\pi}(\pi - \theta) \qquad \text{longitude} = \frac{90}{\pi}\phi$$

Differential Solid Angle

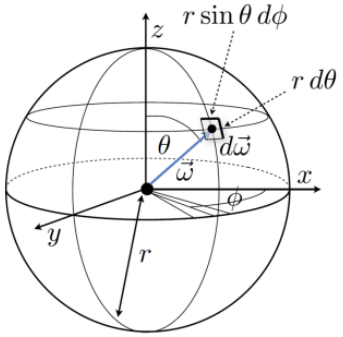
Differential version of an area on a sphere is defined by considering a very small square on the sphere. The differential solid angle can then be defined by dividing the differential area by squared radius of the sphere.

(Note that we denote the differential solid angle with the same symbol as direction.)

Integrating the differential solid angle gives the area of the unit sphere as expected.

$$\begin{aligned}dA &= (r\,d\theta)(r\sin\theta\,d\phi)\\ d\vec{w} &= \frac{dA}{r^2} = \sin\theta\,d\theta d\phi\end{aligned}$$

$$\Omega = \int_S^2 d\vec{w} = \int_0^{2\pi}\int_0^{\pi} \sin\theta\,d\theta\phi = 4\pi$$



5.1.1 Light

Definition 5.1: Light

Light can be a mixture of many wavelengths

Assume **Light** consists of photons each with:

- $\mathbf{x}$: Position
- $\vec{w}$: Direction of Motion
- λ : Wavelength

Each Photon has an energy of: $\frac{hc}{l}$

- $h \approx 6.63 \times 10^{-34} m^2 kg/s$: Planck's constant
- $c = 299,792,458 m/s$: Speed of Light in a vacuum
- With unit Joule $[Joule(J)] = kgm^2/s^2$

We measure light by counting photons.

Spectral Power Distribution (SPD)

- P(λ) = intensity at wavelength λ
- Intensity as a function of wavelength

We perceive these distributions as colours.

5.2 Radiometry

Radiometry is the measurement of electromagnetic radiation, including visible light.
Flux (Radiant Flux, Power)
The total amount of energy passing through a surface or space (A) per unit time.

$$\Phi(A) \qquad \left[\frac{J}{s} = W\right]$$

Examples can include: Number of photons hitting a wall per second. The number of photons leaving a lightbulb per second.

Radiant Intensity

Radiant intensity is defined with differential flux and solid angle. Intuitively, it is a very local measure of energy per unit time per solid angle.

$$I(\vec{w}) = \frac{d\Phi}{d\vec{w}} \qquad \left[\frac{W}{sr}\right] \qquad \int_S^2 I(\vec{w}) d\vec{w}$$

Radiance

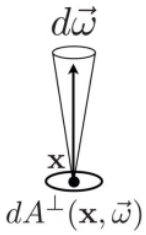
Definition 5.2: Radiance

Radiance is the most important quantity. It is energy per unit time per solid angle per unit **perpendicular** area. Intuitively, due to the cosine term, it is independent of the angle the actual area makes with the light direction.

This makes sense because a large area that is tilted with respect to the light direction does not get more flux.

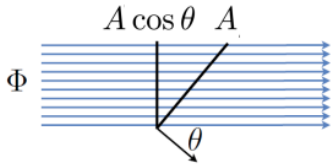
Radiance remains constant along a ray.

We will utilize it in the rendering equation.



$$L(\mathbf{x}, \vec{\omega}) = \frac{d^2\Phi(A)}{d\vec{\omega} dA^{\perp}(\mathbf{x}, \vec{\omega})}$$

$$= \frac{d^2\Phi(A)}{d\vec{\omega} dA(\mathbf{x}) \cos\theta} \quad \left[\frac{W}{m^2 sr}\right]$$



5.3 Reflection Models

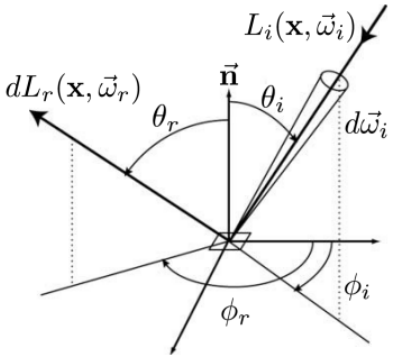
Definition 5.3: BRDF

Bidirectional **R**eflectance **D**istribution **F**unction

BRDF and its variants are the most commonly used functions to represent and render light in a digital scene. With **BRDF**, we only consider reflection, so no light passes through any surface.

Surfaces only reflect

$$f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r) = \frac{dL_r(\mathbf{x}, \vec{w}_r)}{L_i(\mathbf{x}, \vec{w}_i)\cos\theta_i \, d\vec{w}_i}$$



$f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r)$	- Is the **BRDF**
$dL_r(\mathbf{x}, \vec{w}_r)$	- Is an infinitesimal reflected radiance
$d\vec{w}_i$	- Is an infinitesimal solid angle

5.3.1 Reflection and Rendering Equations

Reflection Equation

We can finally define the reflection equation, which is simply the integral form of the previous expression. This tells us that the total reflected light for a given direction is the integral of light coming in all directions on the hemisphere H^2 with weighting terms given by the BRDF and the cosine of the incoming light angle.

(Note: that we do not integrate over the whole sphere as we assume light is only reflected off a surface.)

$$\begin{aligned} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r) &= \frac{dL_r(\mathbf{x}, \vec{w}_r)}{L_i(\mathbf{x}, \vec{w}_i)\cos\theta_i \, d\vec{w}_i} \\ f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r)L_i(\mathbf{x}, \vec{w}_i)\cos\theta_i &= \frac{dL_r(\mathbf{x}, \vec{w}_r)}{d\vec{w}_i} \\ \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r)L_i(\mathbf{x}, \vec{w}_i)\cos\theta_i \, d\vec{w}_i &= L_r(\mathbf{x}, \vec{w}_r) \end{aligned}$$

Definition 5.4: Reflection Equation

The reflected radiance due to incident illumination from all directions

$$L_r(\mathbf{x}, \vec{w}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r)L_i(\mathbf{x}, \vec{w}_i)\cos\theta_i \, d\vec{w}_i$$

Rendering Equation

The step to got from reflectance to rendering equation is adding a term that accounts for potentially emitted light. This ensures that energy is conserved.

Definition 5.5: The Rendering Equation

Outgoing Light = Emitted Light + Reflected Light

$$\begin{aligned} L_o(\mathbf{x}, \vec{w}_o) &= L_e(\mathbf{x}, \vec{w}_o) + L_r(\mathbf{x}, \vec{w}_O) \\ L_o(\mathbf{x}, \vec{w}_o) &= L_e(\mathbf{x}, \vec{w}_o) + \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r)L_i(\mathbf{x}, \vec{w}_i)\cos\theta_i \, d\vec{w}_i \end{aligned}$$

5.4 Illumination

5.4.1 Direct Illumination

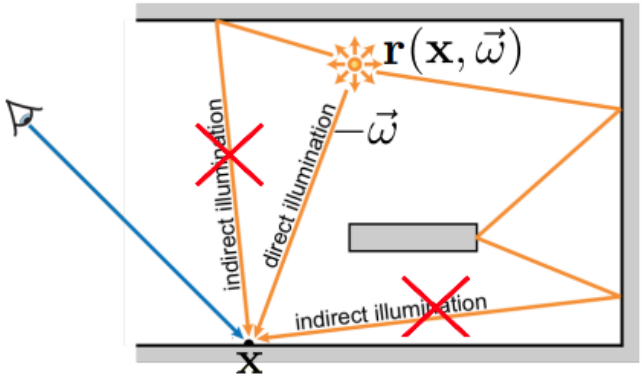
To implement the rendering equation in practice is local or direct illumination.

Definition 5.6: Direct Illumintation

In this model, light can only come from light sources, i.e. we ignore light reflected from a surface and landing on another surface.

This is useful for fast rendering and is what rasterizers such as the one in OpenGL typically assume.

$$L_o(\mathbf{x}, \vec{w}_o) = L_e(\mathbf{x}, \vec{w}_o) + \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r) L_i(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$



$$L_i(\mathbf{x}, \vec{w}) = L_e(\mathbf{r}(\mathbf{x}, \vec{w}), -\vec{w})$$

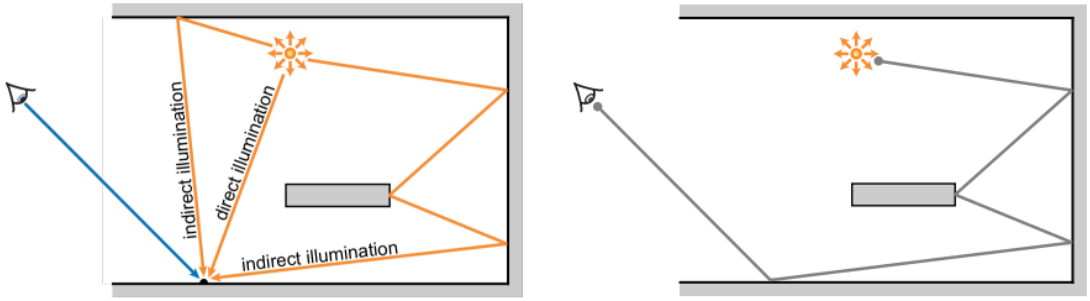
5.4.2 Global Illumination

Definition 5.7: Global Illumination

In general though we should take reflection of light from surfaces into account.

Those are called indirect illumination. This gets quite complicated as light can reflect multiple times before making it to a sensor.

These bounces form a light path.



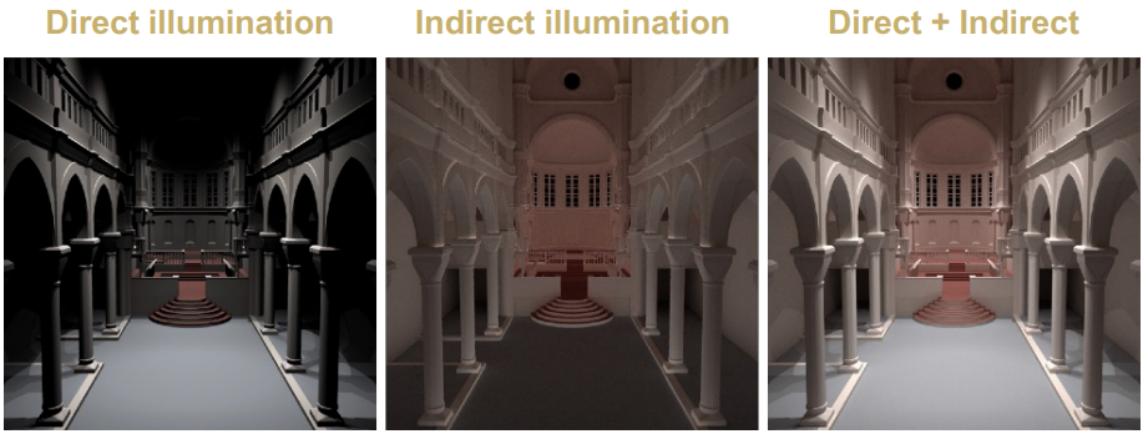
Global Illumination:

- Connects a light source to a sensor sensor.
- Constructed by tracing from:

  - Light Source... Light Tracing
  - From Sensor... Path Tracing
  - Both... Bidirectional Path Tracing

- Length of light path:

  – 2 Segments... Direct Illumination, Direct Lighting
  – >2 Segments... Indirect Illumination, Indirect Lighting



5.5 More on BRDFs

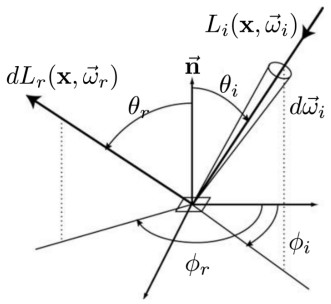
5.5.1 BRDFs - Complex Reflections

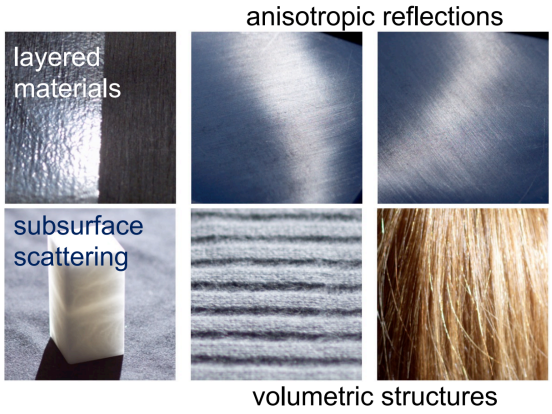
The rendering equation can be used to render any real-world material with complex reflection characteristics.

The BRDF defines the type of reflection for example anisotropic.

If there is non-reflections as in some light passes through, we can define related functions.

$$\int_{H^2} f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_r) L_i(\mathbf{x}, \vec{\omega}_i) \cos\theta_i \, d\vec{\omega}_i$$

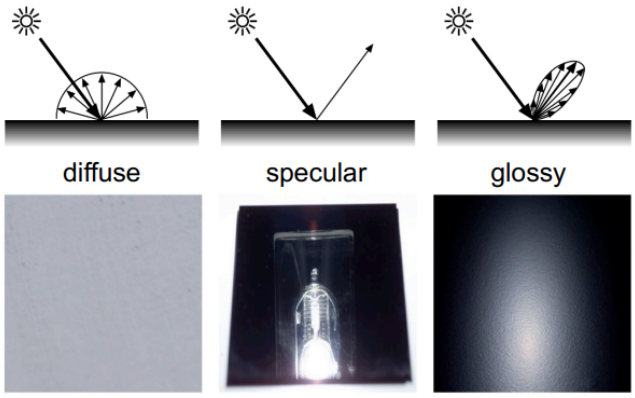




5.5.2 BRDFs - Simpler Reflections

Simple reflection models are useful for fast rendering and can be combined for a wide range of effects.

They result from assuming a special form for the BRDF.



Diffuse

Diffuse reflection means the BRDF is constant and hence can be taken out of the integral. We have a constant times the integral of the incoming light in all directions weighted by the cosine term.

This also means the reflected light is independent of the outgoing light direction, hence no view-dependent effects

$$L_r(\mathbf{x}, \vec{w}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r) L_i(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$

$$L_r(\mathbf{x}) = f_r \int_{H^2} L_i(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$

$$L_r(\mathbf{x}) = f_r E_i(x)$$

6 Distributed Ray Tracing

6.1 Estimating Integrals

We cannot typically compute this integral exactly, so we need to approximate it.

Definition 6.1: Quadrature

Such estimations can be performed by sampling the function to be integrated and summing up the values. This is called **Quadrature**. There is a resulting estimator for the actual value of the integral.

The samples are typically taken from an underlying probability distribution function (PDF). In order to make sure we get the right integral value on average (unbiased estimator), we need to normalize the sampled values.

$$\langle F^N \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

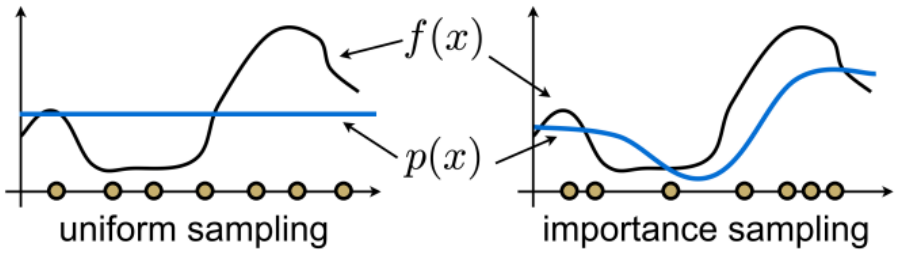
- $\langle F^N \rangle$ Is the estimator.
- N Is the number of samples.
- $f(x_i)$ Is the function to integrate.
- $p(x_i)$ Is the density function.

Applying this idea to the integral we have, we get the following form. The samples are direction samples, i.e. we sample the hemisphere over which we are integrating.

$$L_r(\mathbf{x}, \vec{w}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r) L_i(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$

$$\langle L_r(\mathbf{x}, \vec{w}_r)^N \rangle = \frac{1}{N} \sum_{k=1}^{N} \frac{f_r(\mathbf{x}, w_{i,k}, \vec{w}_r) L_i(\mathbf{x}, \vec{w}_{i,k}) \cos\theta_{i,k} \, d\vec{w}_{i,k}}{p_\Omega(\vec{w}_i, k)}$$

The idea behind importance sampling is simple: place more samples where the function value is high. This is in contrast with uniform sampling, where all samples are uniformly distributed over the domain



Our function to be integrated consists of multiple terms. Ideally, we want to sample their product. But this is typically not possible.

(Note that knowing a function does not mean we can efficiently sample from it.)

The idea is then to choose what to sample. A candidate is the cosine term.

6.2 Importance Sampling

6.2.1 Sampling The Cosine Term

A case where sampling the cosine term is a good idea is when the BRDF f and lighting function L_i are constants.

This is true when we have diffuse objects (constant BRDF) illuminated by the sky (constant L_i).

In this case, we are left with an integral of multiplication of the cosine term and the visibility function.

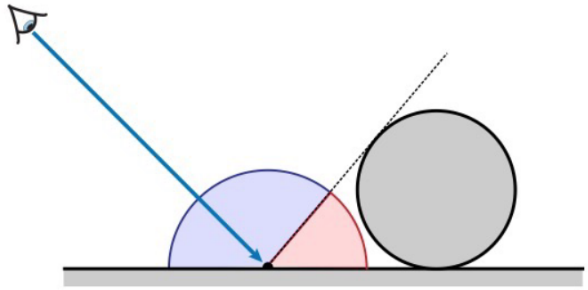
$$L_r(\mathbf{x}, \vec{w}_r) = \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_r) L_i(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$

$$L_r(\mathbf{x}) = \frac{\rho}{\pi} \int_{H^2} V(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$

This is called Ambient Occlusion.

Only some light will reach the point as some is blocked by the object (red region). This detail is captured by the visibility function V, this function is 1 or 0 indicating whether the light can reach the point or not.

For each sample direction, we get a term in the sum. The task is to define what the distribution p is. This is the distribution of the direction samples indexed by k in the sum.



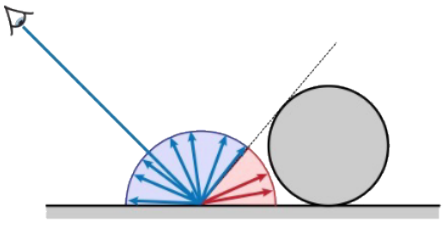
$$L_r(\mathbf{x}) = \frac{\rho}{\pi} \int_{H^2} V(\mathbf{x}, \vec{w}_i) \cos\theta_i \, d\vec{w}_i$$

$$L_r(\mathbf{x}) \approx \frac{\rho}{\pi N} \sum_{k=1}^{N} \frac{V(\mathbf{x}, \vec{w}_{i,k}) \cos\theta_{i,k}}{p(\vec{w}_{i,k})}$$

Uniform sampling means there is no extra weighting by p and we randomly distribute the directions. This is not good as we are wasting samples for which the cosine term is close to zero.

A better approach is importance sampling with the cosine term. This ensures the samples that contribute more have a higher liklihood of being chosen.

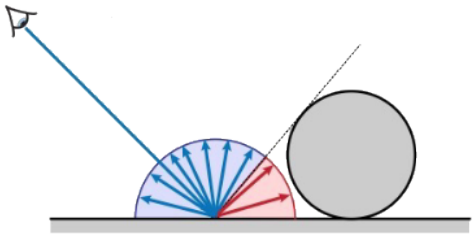
Uniform Sampling



$$p(\vec{w}_{i,k}) = \frac{1}{2\pi}$$

$$L_r(\mathbf{x}) \approx \frac{2\rho}{N} \sum_{k=1}^{N} V(\mathbf{x}, \vec{w}_{i,k}) \cos\theta_{i,k}$$

Importance Sampling



$$p(\vec{w}_{i,k}) = \frac{\cos(\theta_{i,k})}{\pi}$$

$$L_r(\mathbf{x}) \approx \frac{\rho}{N} \sum_{k=1}^{N} V(\mathbf{x}, \vec{w}_{i,k})$$

6.2.2 Sampling BRDFs

In general BRDFs will not be constant and sometimes it is better to importance sample with them. This is typically true for highly peaky functions where BRDF is mostly zero or low except at a few samples. With uniform or cosine based sampling, it is highly probable that we will miss the high values of BRDFs.

$$p(\vec{w}_i) \propto f(\mathbf{x}, \vec{w}_i, \vec{w}_i)$$

6.2.3 Sampling Lights

A further idea is to importance sample the lighting function ($L_i(...)$).

This can also be performed by sampling the light sources in the scene. The advantage of this is that we don't have to sample again for each different point in the scene.

6.3 Global Illumination

So far we considered a local illumination model, light only comes from light sources. In general, light reflects off the surfaces and reaches another surface in the scene. We can thus write the incoming light at a point $\mathbf{x}$ equivalently as the outgoing light at some other point $\mathbf{r}$.

$$L_o(\mathbf{x}, \vec{w}_o) = L_e(\mathbf{x}, \vec{w}_o) + \int_{H^2} f_r(\mathbf{x}, \vec{w}_i, \vec{w}_o) L_i(\mathbf{x}, \vec{w}_i) \cos\theta_i \, \mathrm{d}\vec{w}_i$$

Since we are assuming no participating media then the radiance is constant along rays so we can relate incoming radiance to outgoing radiance:

$$L_i(\mathbf{x}, \vec{w}) = L_o(\mathbf{r}(\mathbf{x}, \vec{w}), -\vec{w})$$

$$L(\mathbf{x}, \vec{w}) = L_e(\mathbf{x}, \vec{w}) + \int_{H^2} f_r(\mathbf{x}, \vec{w}', \vec{w}) L(\mathbf{r}(\mathbf{x}, \vec{w}'), -\vec{w}') \cos\theta' \, \mathrm{d}\vec{w}'$$

To solve this new equation we need to solve the integral and as seen previously that monte carlo methods can be used to estimate integrals of higher dimensions, and since this integral is infinitely dimensional then monte carlo methods are perfect.

Unbiased Methods:

- Recursive ray tracing
- Path tracing and Light tracing
- Bi-directional path tracing

Biased Methods:

- Many-lights algorithm
- Density estimation
- Photon mapping
- Irradiance caching

In a typical rendering framework the BRDF and the cosine term are provided, the problem is the lighting function L, which depends on light coming in from other parts of the scene.

6.4 Recursive Ray Tracing

The solution we obtain is also recursive. At each recursion step, a number of direction samples are taken.

$$L(\mathbf{x}, \vec{w}) = L_e(\mathbf{x}, \vec{w}) + \int_{H^2} f_r(\mathbf{x}, \vec{w}', \vec{w}) L(\mathbf{r}(\mathbf{x}, \vec{w}'), -\vec{w}') \cos\theta' \, \mathrm{d}\vec{w}'$$

$$L(\mathbf{x}, \vec{w}) \approx L_e(\mathbf{x}, \vec{w}) + \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\mathbf{x}, \vec{w}', \vec{w}) L(\mathbf{r}(\mathbf{x}, \vec{w}'), -\vec{w}') \cos\theta'}{p(\vec{w}')}$$

We send a random ray from the camera into the scene, this gives us the point we want to sample the amount of light we receive in this direction.

We then sample the hemisphere using either uniform or importance sampling to sample the light incoming to this point from the rest of the scene.
Some of these rays will hit light sources which we can easily calculate the lighting function L and so we can stop this direction.
Others will hit another point in the scene that is not a light so we need to repeat this step to estimate L for this point.

This algorithm is highly inefficient as we increase the depth, d, the number of ray samples we need to evaluate increases exponentially, where n is the number of ray samples we send per intersecion.

$$\#\text{ray samples} \in O(n^d)$$

6.4.1 Shadow Ray Optimisation

A shadow ray is simply a ray that connects a point in the scene to a light source. The contribution of light incoming from this sampled light is then estimated by this shadow ray. Any other sample rays that we send from **this** point that intersect this light source we terminate and discard them in order to avoid double counting the lights contribution.

6.5 Path Tracing

Since as seen in the previous subsection the running time of Recursive Ray Tracing is exponential it is not used in practice. What is is something called path tracing, the difference between Path Tracing and Recursive Ray Tracing is after we send the first ray into the scene we only sample one direction and trace this, and do the same at the next intersection point, we repeat until we reach a light source, or in some cases exceed the maximum depth that rays are allowed to travel.

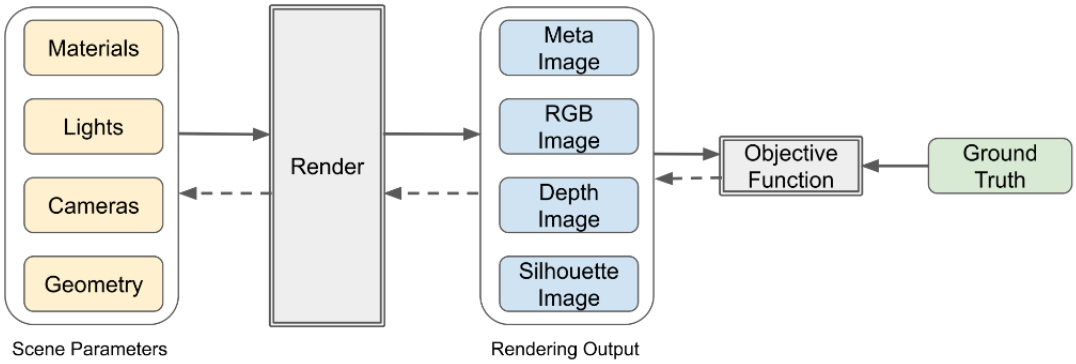
This method is very simple to implement and has a large collection of optimisations that can be implemented.

However, Path Tracing is slow to converge as it requires $4\times$ more samples to half the error

Path Tracing also has robustness issues i.e. it does not handle some light paths well such as caustics, there is also no reuse of caching or computation.

7 Inverse Rendering

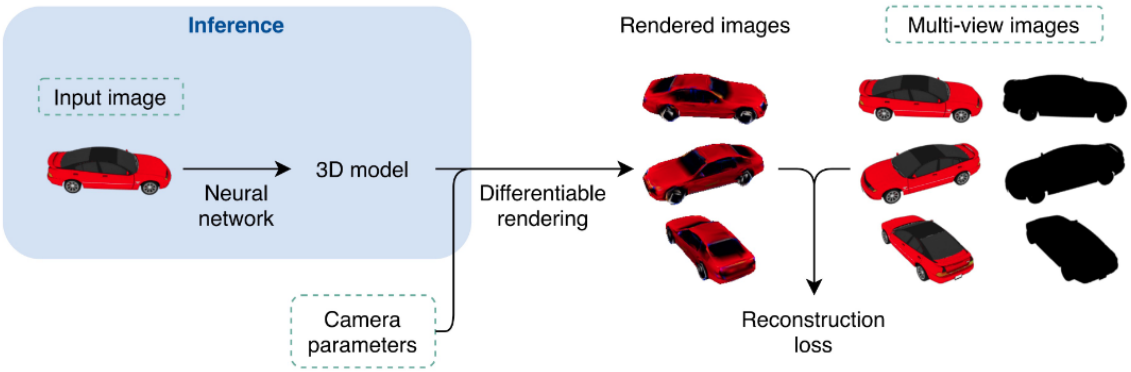
Graphics is traditionally about forward rendering. That is what we have seen so far, going from scenes to their renderings. A recent problem in graphics is the inverse of this, going from images to scenes via inverse rendering.



Forward rendering means taking a scene definition with geometry, materials, cameras and lights and then generating an image from this data.

Inverse rendering starts from the rendered image, compares it to some form of ground truth image, and updates the parameters of the scene elements with the loss that compares the images.

An application of this is geometry reconstruction from a single image. At testing (inference) time, we can input an image into a neural network and get the corresponding 3D model. This network is optimised via rendering the 3D model generated from an image and comparing the rendered images with the ground truth images.



Inverse rendering requires a non-linear optimisation: the function from the scene parameters to rendered images is highly non-linear. Deep learning frameworks help us with this optimisation, e.g. from stochastic gradient descent. There is one essential requirement for this to work, we need differential rendering.

Everything in the rendering equation is differentiable except for the visibility term. This is because it is a binary function either 0, or 1.

This means that the function we integrate is discontinuos. This means we cannot exchange the order of derivates with the integral.

$$\frac{\partial}{\partial \pi} \int_A f(\mathbf{x}) \mathrm{d}\mathbf{x} \neq \int_A \frac{\partial}{\partial \pi} f(\mathbf{x}) \mathrm{d}\mathbf{x}$$