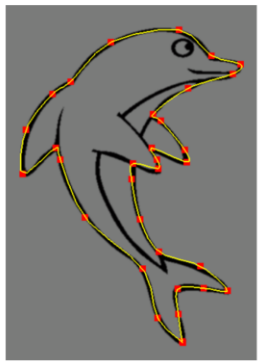
”Out of nothing I have created a strange new universe.”

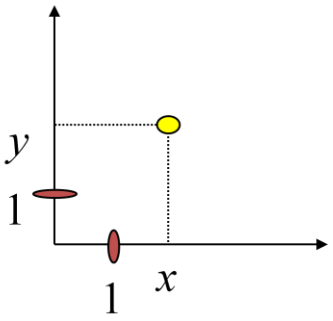
Bolyai János

Geometric Modeling Points and Curves

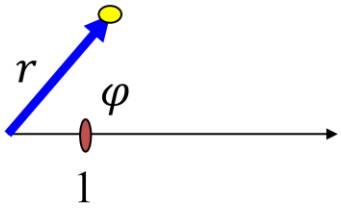
Szirmay-Kalos László



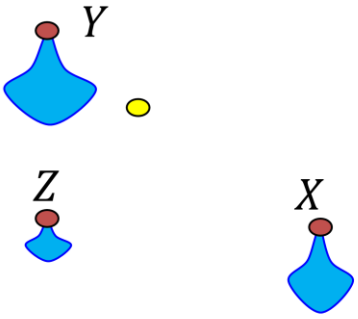
Point definition with coordinate systems



Descartes



Polar



Barycentric
Homogeneous

With numbers!

1. Coordinate system (=reference geometry)
2. Coordinates (=measurement)

The goal is the definition of points with numbers and primitives with equations or functions.

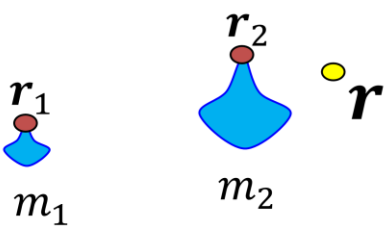
The definition of points with numbers requires a **coordinate system** and then the **measuring of the point with respect to this coordinate system**. A **Cartesian coordinate system** contains two orthogonal lines or axes, and a unit on them, and we measure how far we should walk along them to arrive at the identified point.

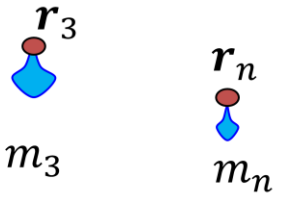
A 2D **polar coordinate system** is a half line, and a point is defined by an angle and a distance. The angle specifies the direction in which we should go from the origin and the distance is interpreted between the origin and the identified point. Note that while we require that all points can be expressed by coordinates, this is not necessarily unambiguous, i.e. in a polar system the origin can be defined with arbitrary angle and with distance zero.

In computer graphics **barycentric coordinate systems** are also popular. Here, the coordinate system is a set of points (at least 3 in 2D) where we put weights. The resulting mechanical system has a center of mass somewhere, which are identified by the numbers of the weights. Barycentric coordinates are often called **homogeneous**, due to the property that if we multiply all weights with the same non-zero scalar, then the center of mass is not affected.

However, for such constructions we have already applied many non-trivial concepts like vectors, distance, angles. First, let us start from scratch and revisit these basic building blocks.

Barycentric (homogeneous) coordinates





Torque is zero

$$\sum_i (\boldsymbol{r}_i - \boldsymbol{r}) \times m_i \boldsymbol{g} = 0$$

$$\boldsymbol{r} = \frac{\sum_i m_i \boldsymbol{r}_i}{\sum_i m_i}$$

- $\boldsymbol{r}$ is the combination of points $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n$
- If weights are not negative: **convex combination**
- Convex combination is in the convex hull
- Line (segment) = Combination (convex) of 2 points
- Plane (triangle) = Combination (convex) of 3 points

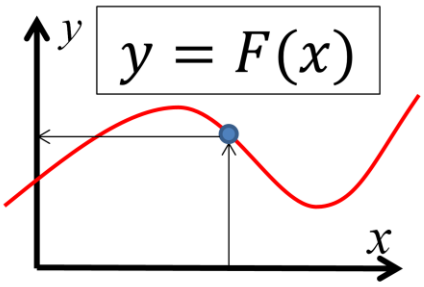
Defining a point as the center of mass of a system where masses placed at finite number of reference points is also called the **combination of these points with barycentric coordinates** equal to the weights.

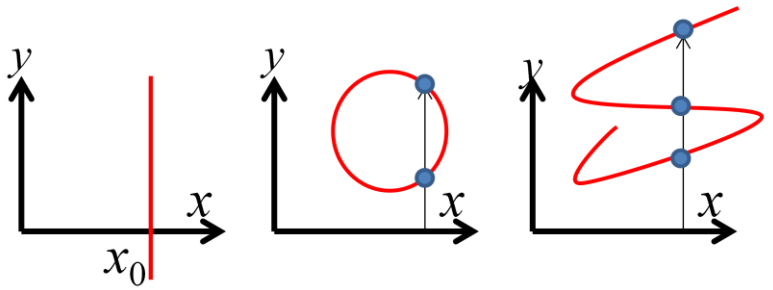
Note that we can do this in real life without mathematics and coordinate systems, center mass exists and is real without mathematics and abstraction.

If all weights are non-negative, which has direct physical meaning, then we talk of **convex combination** since the points that can be defined in this way are in the convex hull of the reference points. By definition, the **convex hull** is the minimal set of points that is convex and includes the original reference points. For example, when presents are wrapped, the wrapping paper is on the convex hull of the presents.

Using the term combination or convex combination, we can define a line as a combination of two points and a line segment as a convex combination of two points. Similarly, the convex combination of three not collinear points is the triangle, the convex combination of four points not being in the same plane is a tetrahedron.

Curves: Explicit equation





2D line:

$y = mx + b$

Not good:

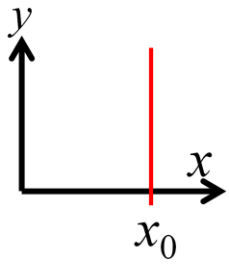
For one x, we have not exactly one y

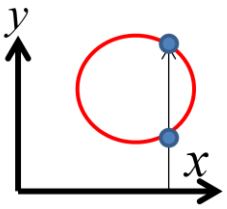
If we want to specify 1D objects, like curves, then we should simultaneously identify (uncountably) infinitely many points. Obviously, defining the points one by one with their Cartesian coordinates is not an option. Instead, we usually specify an equation that has infinitely many roots and these roots are considered as the Cartesian coordinates of points in a set defined by the equation. Assume that we are in 2D when the equation should contain Cartesian coordinates x and y (in 3D there would be a third coordinate as well).

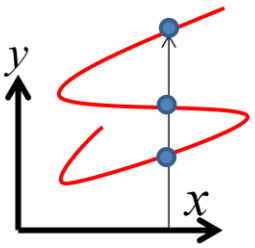
The most obvious, but the least useful equation type is the **explicit form**, where we express y as a function of x. The problem with this representation is for each x there must be exactly one y. This is usually not the case, think of a circle or a vertical line, for example.

Curves: Implicit equation

$$f(x,y) = 0 \text{ or } f(\boldsymbol{r}) = 0$$







2D line:

$$ax + by + c = 0$$

Circle:

$$(x - x_0)^2 + (y - y_0)^2 - R^2 = 0$$

The equation can have **implicit** form, which means that x and y are put into an algebraic expression that is made equal to zero. We have a single equation with two unknowns, thus, we have the hope of having infinitely many roots, i.e. x,y pairs.

For example, a linear equation containing x and y identifies a line. A circle contains points that are at distance R from the reference point. Expressing this distance with the Pythagoras theorem, we can also develop and equation for the circle.

Curves: Parametric equation

$$x = x(t), y = y(t), z = z(t)$$
or $\boldsymbol{r} = \boldsymbol{r}(t)$

3D line:

$$x(t) = x_0 + v_x\, t$$
$$y(t) = y_0 + v_y\, t$$
$$z(t) = z_0 + v_z\, t$$

$t \in (-\infty, \infty)$

Circle:

$$x(t) = x_0 + R\cos(2\pi t)$$
$$y(t) = y_0 + R\sin(2\pi t)$$

$t \in [0,1)$

The curve equation may also have **parametric form**, where we use a free parameter t that can run in an appropriate interval.

Substituting t into two equations defining x and y (or z), we get the Cartesian coordinates of the point corresponding to t.

Parameter t can be imagined as time and the parametric function as the definition of a motion or path.

Zoo of Classical Curves

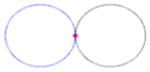
Parabola: $|\boldsymbol{r} - \boldsymbol{f}| = |\boldsymbol{n}^0 \cdot (\boldsymbol{r} - \boldsymbol{r}_0)|$

Ellipse: $|\boldsymbol{r} - \boldsymbol{f}_1| + |\boldsymbol{r} - \boldsymbol{f}_2| = C$

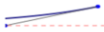
Hyperbola: $|\boldsymbol{r} - \boldsymbol{f}_1| - |\boldsymbol{r} - \boldsymbol{f}_2| = C$

Cycloid:
$$x(t) = t - \sin t$$
$$y(t) = 1 - \cos t$$

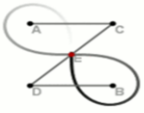
Cardioid: $(x^2 + y^2)^2 - 2ax(x^2 + y^2) = a^2y^2$



Tractrix:
$$x(t) = \operatorname{sech} t$$
$$y(t) = t - \tanh t$$

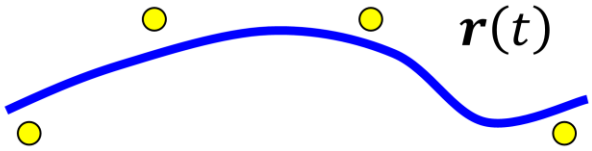


Lemniscate: $(x^2 + y^2)^2 = 2c^2(x^2 - y^2)$



Free form curves

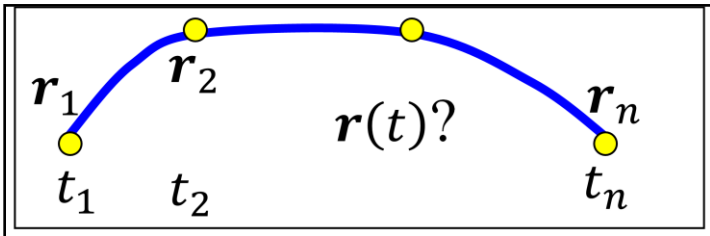
- Definition with control points



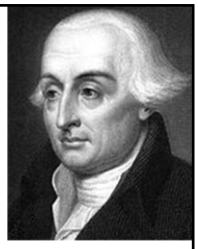
- Polynomial: $x(t) = \sum_i a_i t^i, y(t) = \sum_i b_i t^i, z(t) = \cdots$
- Polynomial coefficients: A curve should be
  - close to the control points: **Interpolation/Approximation**
  - **natural:** C^2 continuity
  - **beautiful**: close to constant curvature without waves
  - independent of the coordinate system (center of mass)
  - locally controllable

Curves we meet usually do not belong to the category of classic curves, so we do not know their equation. These curves are **free form curves**.

As ”everything” can be approximated by polynomials, the unknown equations of free form curves are also attacked this way. We approximate their parametric equations with polynomials of parameter t. The problem is that the polynomial coefficients do not have intuitive interpretation, thus we cannot expect the modeler to specify the coefficients directly. Instead, we require the user to specify a finite number of **control points**, and the modeling program automatically computes the polynomial coefficients from the control points. This computation can be an **interpolation** when the resulting curve is expected to go through the control points. Or, the computation can also be an **approximation**, when the resulting curve should just somehow follow the control points, but it does not have to go through each of them. By requiring approximation instead of interpolation, we ease the fitting process so we can impose additional requirements concerning the ”quality” of the curve.



Lagrange interpolation



- Find: $\boldsymbol{r}(t) = \left(\sum_i a_i t^i,\ \sum_i b_i t^i,\ \sum_i c_i t^i\right)$, where $\boldsymbol{r}(t_1) = \boldsymbol{r}_1,\ \boldsymbol{r}(t_2) = \boldsymbol{r}_2,\ \ldots,\ \boldsymbol{r}(t_n) = \boldsymbol{r}_n$
- Degree of the polynomial? $n - 1$

$$\boldsymbol{r}(t) = \sum_i L_i(t)\boldsymbol{r}_i \quad \rightarrow \quad \boldsymbol{r}(t_k) = \sum_i L_i(t_k)\boldsymbol{r}_i = \boldsymbol{r}_k$$

$$L_i(t) = \frac{\prod_{j\neq i}(t - t_j)}{\prod_{j\neq i}(t_i - t_j)}$$

$$L_i(t_k) = \frac{\prod_{j\neq i}(t_k - t_j)}{\prod_{j\neq i}(t_i - t_j)} \begin{cases} 1 \text{ if } i = k \\ 0 \text{ if } i \neq k \end{cases}$$

The first curve is of interpolation type and is known as the **Lagrange interpolation**. Suppose we specify a sequence of control points r1, …, rn, and search a parametric function r(t) (one polynomial for each of the x, y or x, y, z coordinates) that goes through them. More precisely, we expect the curve to give control point r1 for parameter value t1, r2 for t2, etc. The interpolation requirement means n constraints, thus the polynomials may have n unknown coefficients to make the number of unknowns equal to the number of equations, and thus obtaining a well defined problem with an unambiguous solution. To find the n unknown polynomial coefficients, we need to solve a linear equation generated by substituting t1,…,tn into the polynomial and requiring them to be equal to r1,…,rn, respectively. If we solve it, we obtain the coefficients, which allow the computation of the curve point for arbitrary parameter t.

Instead of solving the linear equation, the solution can be given directly as a combination of the control points with barycentric coordinates Li(t) that depend on parameter t. The algebraic form of these weight functions, aka **basis functions** or **blending functions** is shown here as the ratio of two products.

To prove that the combination of the control points with these functions satisfies the interpolation constraints, let us examine a basis function Li when we substitute tk into it. If i=k, the numerator and the denominator of Li will be similar, so Li(ti) = 1. However, when i !=k, there will be some j which equals to k, so one factor of the numerator will be tk-tk=0, making Li(tk) also zero. So Li is 1 for ti but is zero for all other discrete parameter values. This means that in sum Li(tk) ri, all control points ri get zero weight except rk, which gets weight 1, thus r(tk) = rk.

Note: A point of the Lagrange curve is the combination of control points with weights Li. According to the definition of combination, the reference points (which are the control points here) should be multiplied with the corresponding weights, the terms should be

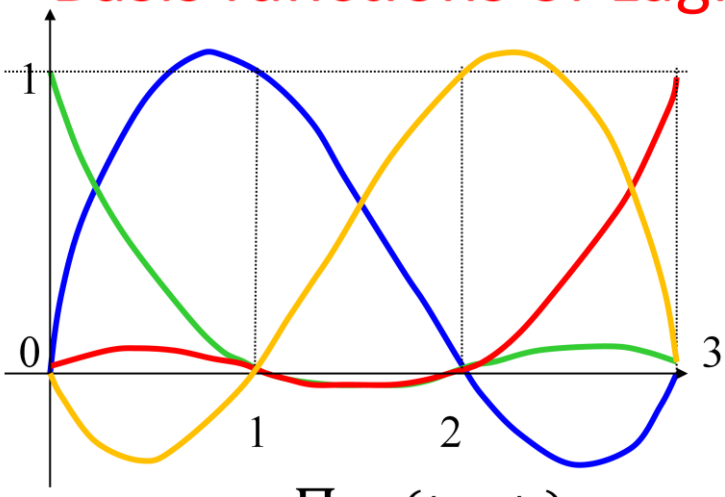
added them up, and finally the sum be divided with the total mass. Where is this division? The division can be ignored if the total mass is 1. Is the sum of the weight functions equal to 1 for any t??? (Yes).

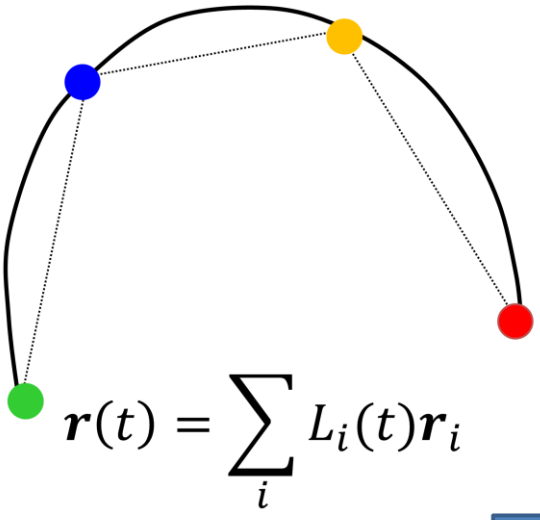
```
class LagrangeCurve {
   vector<vec3>  cps; // control pts
   vector<float> ts;  // knots
   float L(int i, float t) {
        float Li = 1.0f;
        for(int j = 0; j < cps.size(); j++)
               if (j != i) Li *= (t - ts[j])/(ts[i] - ts[j]);
        return Li;
   }
public:
   void AddControlPoint(vec3 cp) {
        float ti = cps.size();        // or something better
        cps.push_back(cp); ts.push_back(ti);
   }
    vec3 r(float t) {
        vec3 rt(0, 0, 0);
        for(int i=; i < cps.size(); i++) rt += cps[i] * L(i,t);
        return rt;
   }
};
```

$$L_i(t) = \frac{\prod_{j \neq i}(t - t_j)}{\prod_{j \neq i}(t_i - t_j)}$$

$$\boldsymbol{r}(t) = \sum_i L_i(t)\boldsymbol{r}_i$$

Basis functions of Lagrange interpolation





$$L_i(t) = \frac{\prod_{j \neq i}(t - t_j)}{\prod_{j \neq i}(t_i - t_j)}$$

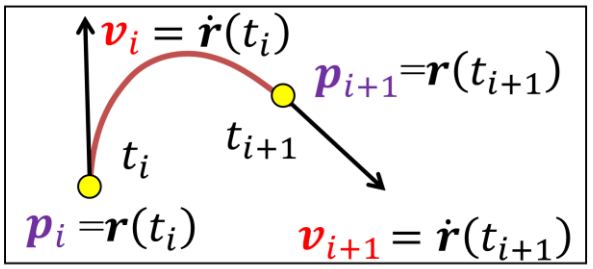
$$\boldsymbol{r}(t) = \sum_i L_i(t)\boldsymbol{r}_i$$

Let us take an example where there are four control points and we expect the curve to interpolate them for t=0, 0.33, 0.67, and 1, respectively. The basis functions are depicted in the Figure. When t=0, the weight of the green point is 1, and the weight of all other points is zero. The curve is then in the green point. When t increases, the weight of the red point gets larger and at t=0.33 only the red point has non-zero weight…

The basis functions **oscillate** between positive and negative values, thus a control point periodically attracts or repels the curve. This is bad since the curve will tend to oscillate.

The other disadvantage of Lagrange interpolation is that it cannot provide **local control**. Local control would mean that the modification of a control point modifies only a smaller part of the curve. However, as all basis functions are non-zero in the whole domain, the complete curve will change.

(Charles) Hermite interpolation



$$a_0 = p_i$$
$$a_1 = v_i$$
$$a_2 = \frac{3(p_{i+1} - p_i)}{(t_{i+1} - t_i)^2} - \frac{(v_{i+1} + 2v_i)}{t_{i+1} - t_i}$$
$$a_3 = \frac{2(p_i - p_{i+1})}{(t_{i+1} - t_i)^3} + \frac{(v_{i+1} + v_i)}{(t_{i+1} - t_i)^2}$$

- $r(t) = a_3(t - t_i)^3 + a_2(t - t_i)^2 + a_1(t - t_i) + a_0$
- $\dot{r}(t) = 3a_3(t - t_i)^2 + 2a_2(t - t_i) + a_1$

$$r(t_i) = a_0 = p_i$$
$$r(t_{i+1}) = a_3(t_{i+1} - t_i)^3 + a_2(t_{i+1} - t_i)^2 + a_1(t_{i+1} - t_i) + a_0 = p_{i+1}$$
$$\dot{r}(t_i) = a_1 = v_i$$
$$\dot{r}(t_{i+1}) = 3a_3(t_{i+1} - t_i)^2 + 2a_2(t_{i+1} - t_i) + a_1 = v_{i+1}$$

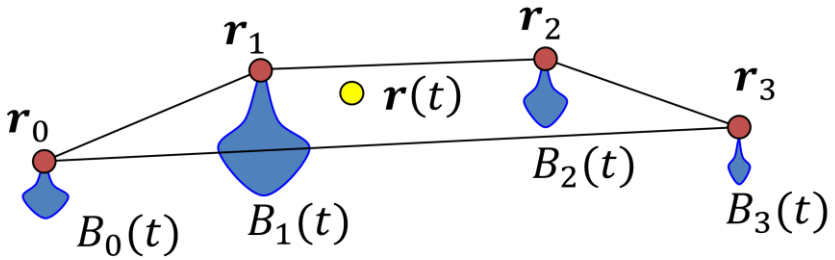
Hermite (H at the beginning and e at the end are silent because he was a Frenchman) interpolation is a generalization of Lagrange interpolation, where not only the points to be interpolated are given but also the derivatives. Here we discuss only the practically relevant special case, when the curve is defined by two control points and the first derivatives at these control points. We have four constraints, so the polynomial that is unambiguously determined by these constraints if a cubic (of four polynomical coefficients).

The strategy is (always) similar to that of the Lagrange interpolation. We take the polynomial with yet unknown coefficients, substitute the constraints, and get a linear equation for the unknown coefficients. This linear equation is solved.

(Pierre) Bézier approximation

Find: $\boldsymbol{r}(t) = \sum_i B_i(t)\boldsymbol{r}_i$

- $B_i(t)$: no oscillation
- inside the convex hull
- $B_i(t) \geq 0, \quad \sum_i B_i(t) = 1$



Lagrange (and Hermite) interpolation tends to oscillate. Let us find a better curve. We still use the center of mass analogy, i.e. the curve will be the composition of control points with weights placed at them. The weights are basis functions Bi(t) and we can ignore division with the total mass if the sum of weights is guaranteed to be equal to 1.

We do not want the oscillation of the Lagrange curve, so we allow only non-negative weights. Composition with non-negative weights is a convex combination, thus all points of the curve, i.e. the complete curve will be in the convex hull of the control points.

(Серге́й Ната́нович) Bernstein polynomials

Newton's binomial theorem

$$1^n = (t + (1-t))^n = \sum_{i=0}^{n} \underbrace{\binom{n}{i} t^i (1-t)^{n-i}}_{B_i(t)}$$

$$B_i(t) \geq 0,\ \sum_i B_i(t) = 1\ :\ \text{OK}$$

So, the task is to find a basis function system where each basis function is non-negative in the allowed domain (in [0,1]) and their sum is everywhere 1.

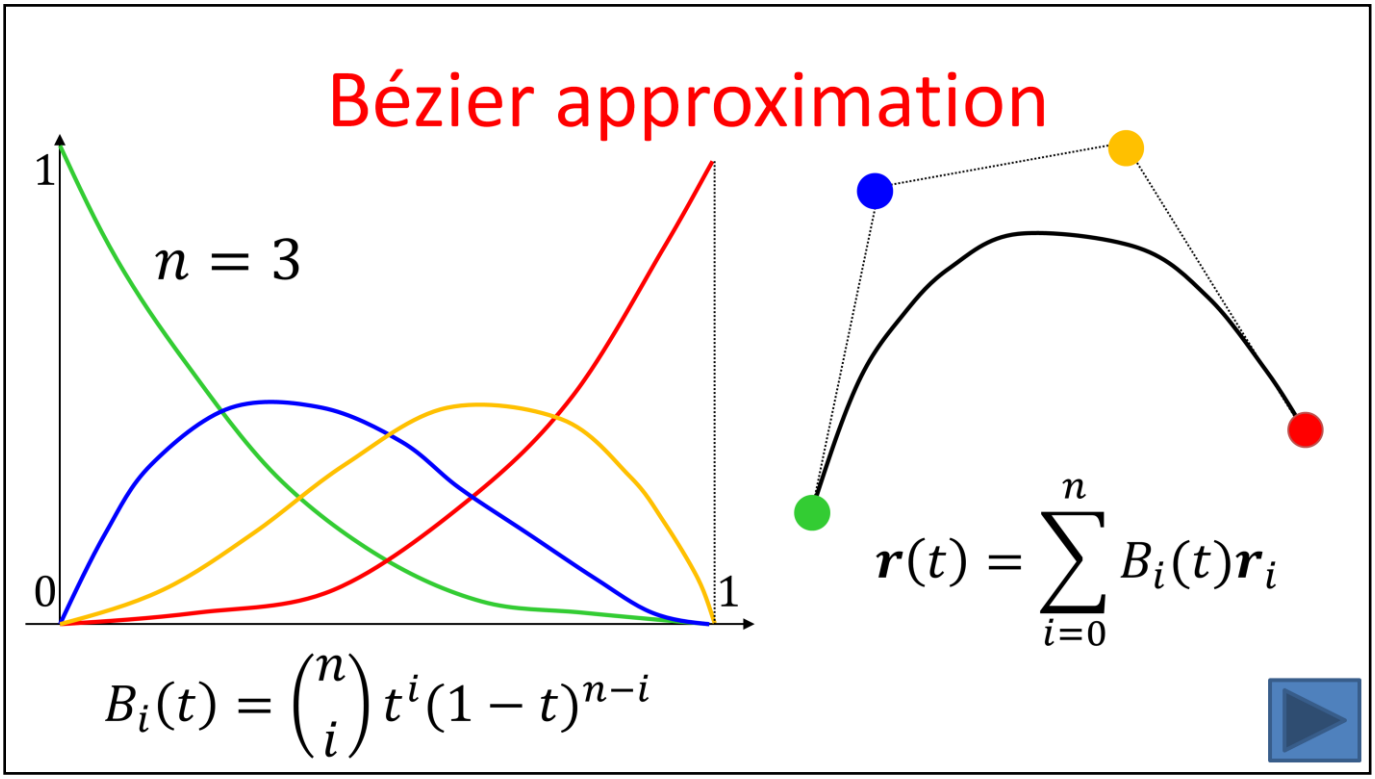
Such basis functions can be constructed by expressing 1 with the Newtonian binomial theorem. The terms are called Bernstein polynomials, which are indeed non-negative if t is in [0,1], and their creation guarantees that their sum is 1.

```
class BezierCurve {
   vector<vec3> cps;// control pts
   float B(int i, float t) {
       int n = cps.size()-1; // n deg polynomial = n+1 pts!
       float choose = 1;
       for(int j = 1; j <= i; j++) choose *= (float)(n-j+1)/j;
       return choose * pow(t, i) * pow(1-t, n-i);
   }
public:
   void AddControlPoint(vec3 cp) { cps.push_back(cp); }

   vec3 r(float t) {
       vec3 rt(0, 0, 0);
       for(int i=0; i < cps.size(); i++) rt += cps[i] * B(i,t);
       return rt;
   }
};
```

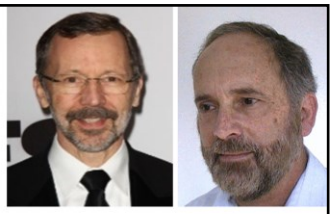
$$B_i(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\boldsymbol{r}(t) = \sum_{i=0}^{n} B_i(t)\boldsymbol{r}_i$$

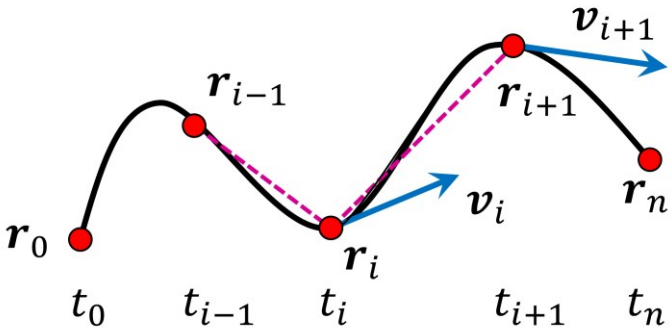


If n = 3 (which is good for 4 control points), the basis functions are (1-t)^3, 3*(1-t)^2*t, 3*(1-t) *t^2, t^3. Note that the first basis function is 1 for t=0, while all others are zero, so the curve crosses the first control points. Similarly, when t=1, the curve is at the last control point. However, other control points are not so lucky, they are usually not interpolated. This is an approximation curve.

Catmull-Rom spline



- A Hermite between every two consecutive points
- C^1: velocity must be shared
- Approximate C^2: The shared velocity is selected to make the acceleration approximately continuous



$$v_i = \frac{1}{2}\left(\frac{r_{i+1} - r_i}{t_{i+1} - t_i} + \frac{r_i - r_{i-1}}{t_i - t_{i-1}}\right)$$

Let us define a separate curve segments between every two control points applying Hermite interpolation. Hermite interpolation needs the start and end points (which are available) and the derivatives at these two points (which should be found somehow).

If the speed is uniform and the motion is linear in segment i, then its constant speed equals to (r{i}-r{i-1})/(t{i}-t{i-1}).

Similarly the constant speed in segment i+1 would be (r{i+1}-r{i})/(t{i+1}-t{i}). A good approximation is to set the velocity at the control point shared by the two segments to the average of these two velocities. This is the Catmull-Rom spline.

Kochanek and Bartels further generalized this spline and allowed an additional tension parameter that can scale up or down the average velocity. On the other hand, we can use a weighted average when the average of the two constant speeds is obtained.

```
class CatmullRom {
  vector<vec3>  cps; // control points
  vector<float> ts;   // parameter (knot) values
  vec3 Hermite(vec3 p0, vec3 v0, float t0,
               vec3 p1, vec3 v1, float t1, float t ) {
  }
public:
  void AddCtrlPoint(vec3 cp, float t) {…}
  vec3 r(float t) {
    for(int i = 0; i < cps.size() - 1; i++)
      if (ts[i] <= t && t <= ts[i+1]) {
        vec3 v0 = …, v1 = …;
        return Hermite(cps[i],v0,ts[i], cps[i+1],v1,ts[i+1], t);
      }
  }
};
```

$$r(t) = a_3(t-t_0)^3 + a_2(t-t_0)^2 + a_1(t-t_0) + a_0$$

$$a_0 = p_i\ ,\ a_1 = v_i$$

$$a_2 = \frac{3(p_{i+1}-p_i)}{(t_{i+1}-t_i)^2} - \frac{(v_{i+1}+2v_i)}{t_{i+1}-t_i}$$

$$a_3 = \frac{2(p_i-p_{i+1})}{(t_{i+1}-t_i)^3} + \frac{(v_{i+1}+v_i)}{(t_{i+1}-t_i)^2}$$

$$v_i = \frac{1}{2}\left(\frac{r_{i+1}-r_i}{t_{i+1}-t_i} + \frac{r_i-r_{i-1}}{t_i-t_{i-1}}\right)$$

The Catmull-Rom spline can be found in PowerPoint and in many drawing packages. It is an interpolating spline with **local control**.

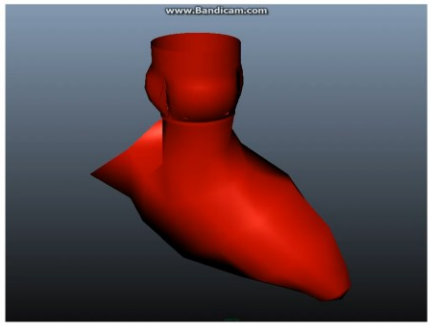
When we move a control point, the average speeds of two linear uniform motions are modified. Thus, the averages of these linear motions are changed at three control points, which can affect four curve segments at most.

”La semplicità è la sofisticazione finale.”

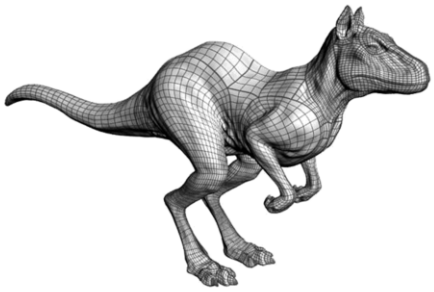
Leonardo da Vinci

Geometric Modeling Surfaces

Szirmay-Kalos László



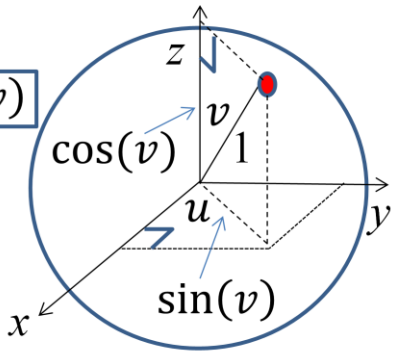
Surfaces



Surface is a 2D manifold of the 3D space:

- **Explicit:** $z = h(x, y)$
- **Implicit**: $f(x, y, z) = 0$
  - sphere: $(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - R^2 = 0$
  - plane: $ax + by + cz + d = 0$
- **Parametric**: $x = x(u, v), y = y(u, v), z = z(u, v)$
  - sphere:

$$x(u, v) = c_x + R\cos(u)\sin(v)$$
$$y(u, v) = c_y + R\sin(u)\sin(v)$$
$$z(u, v) = c_z + R\cos(v)$$
$$u \in [0, 2\pi), v \in [0, \pi)$$



Surfaces are two-dimensional subsets of the 3D space. Their definition is very similar to that of curves, but now the parametric equations have two free parameters (parametric equations of curves map a line segment onto the curve, parametric equations of surfaces map a square onto the surface).

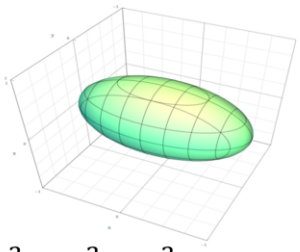
Implicit quadratic surfaces

- **Sphere:** Points $\boldsymbol{r}(x,y)$ that are at distance R from center $\boldsymbol{c}(c_x,c_y)$: $|\boldsymbol{r}-\boldsymbol{c}|=R$
- **Ellipsoid:** Points $\boldsymbol{r}$ from where the sum of distances to focal points $\boldsymbol{f}_1,\boldsymbol{f}_2$ is constant C: $|\boldsymbol{r}-\boldsymbol{f}_1|+|\boldsymbol{r}-\boldsymbol{f}_2|=C$
- **Hyperboloid**: Points $\boldsymbol{r}$ from where the difference of distances to focal points $\boldsymbol{f}_1,\boldsymbol{f}_2$ is constant C: $|\boldsymbol{r}-\boldsymbol{f}_1|-|\boldsymbol{r}-\boldsymbol{f}_2|=C$
- **Paraboloid:** Set of points $\boldsymbol{r}$ that are at the same distance from focal point $\boldsymbol{f}$ as from plane of normal vector $\boldsymbol{n}$ and position vector $\boldsymbol{p}$: $|\boldsymbol{r}-\boldsymbol{f}|=|\boldsymbol{n}^0\cdot(\boldsymbol{r}-\boldsymbol{p})|$

Implicit: Quadrics

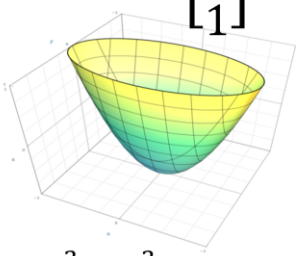
$$f(x,y,z) = [x,y,z,1]\mathbf{Q}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

$\mathbf{Q}$ can be made symmetric



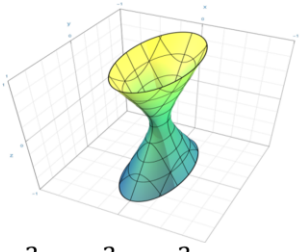
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}-1=0$$

Ellipsoid



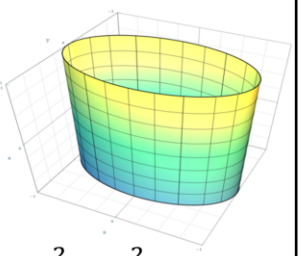
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-z=0$$

Paraboloid



$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}-1=0$$

Hyperboloid



$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-1=0$$

Elliptic Cylinder

Normal vector of implicit surfaces

- Surface points: $f(x^*, y^*, z^*) = 0$
- Normal vector $= \nabla f(x^*, y^*, z^*)$

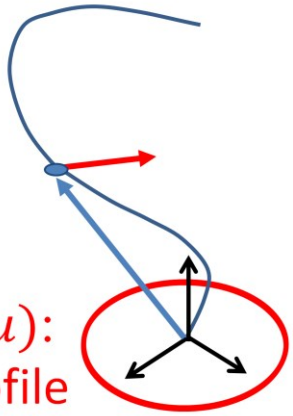
$$
\begin{aligned}
0 &= f(x, y, z) \\
&= f(x^* + (x - x^*), y^* + (y - y^*), z^* + (z - z^*)) \\
&\approx f(x^*, y^*, z^*) + \frac{\partial f}{\partial x}(x - x^*) + \frac{\partial f}{\partial y}(y - y^*) + \frac{\partial f}{\partial z}(z - z^*) \\
&= \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) \cdot (x - x^*, y - y^*, z - z^*) = 0
\end{aligned}
$$

$$\boldsymbol{n} \cdot (\boldsymbol{r} - \boldsymbol{p}) = 0$$

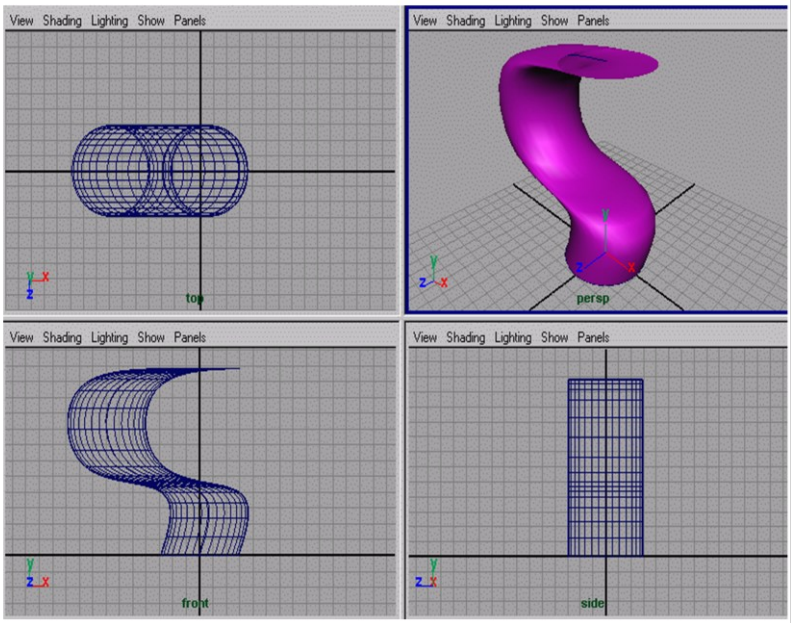
Parametric surfaces: Extruding

$$\boldsymbol{r}(u,v) = \boldsymbol{s}(v) + \boldsymbol{b}(u)$$

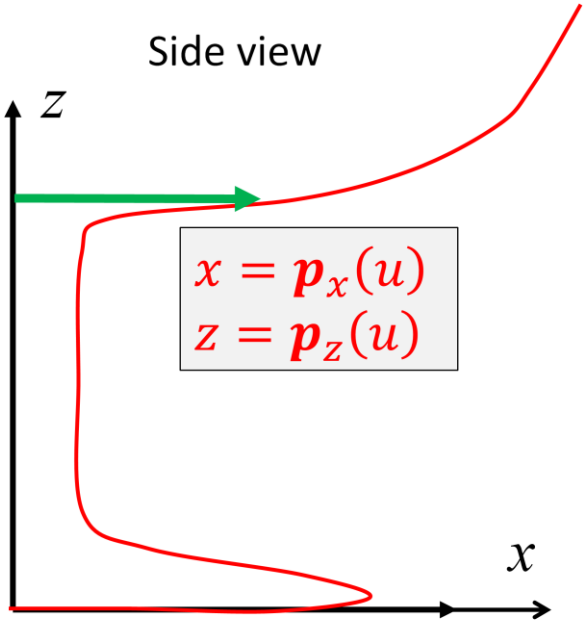
$\boldsymbol{s}(v)$: spine

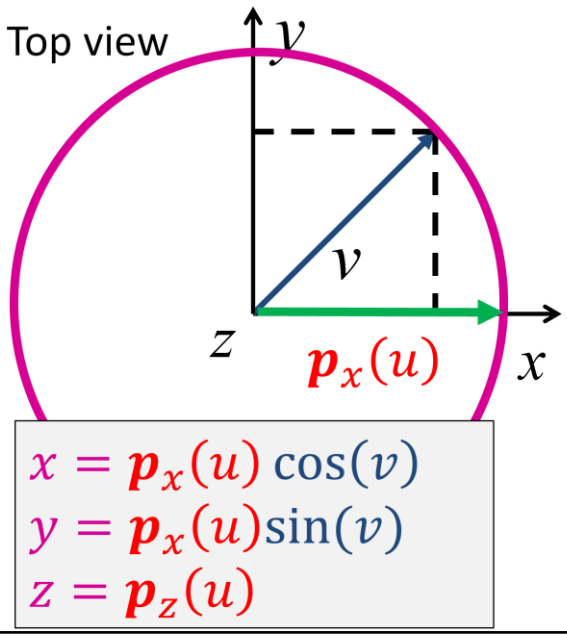


$\boldsymbol{b}(u)$: profile



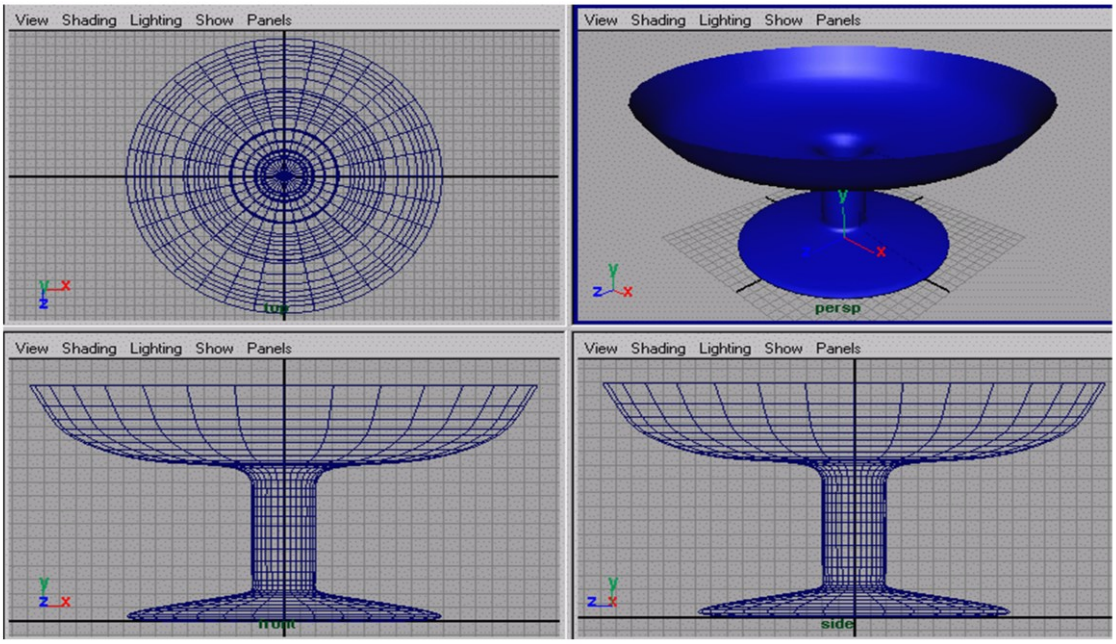
Parametric surfaces: Rotating



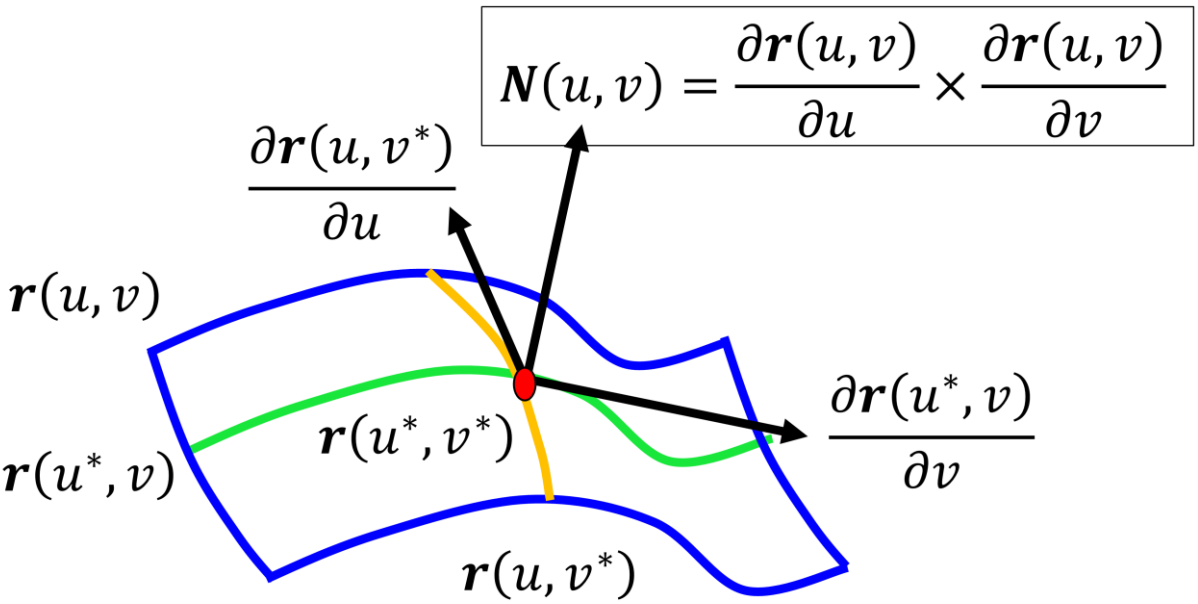


$x = \boldsymbol{p}_x(u)\cos(v)$
$y = \boldsymbol{p}_x(u)\sin(v)$
$z = \boldsymbol{p}_z(u)$

Rotating


View Shading Lighting Show Panels
top
View Shading Lighting Show Panels
persp
View Shading Lighting Show Panels
front
View Shading Lighting Show Panels
side

Normal vector of parametric surfaces



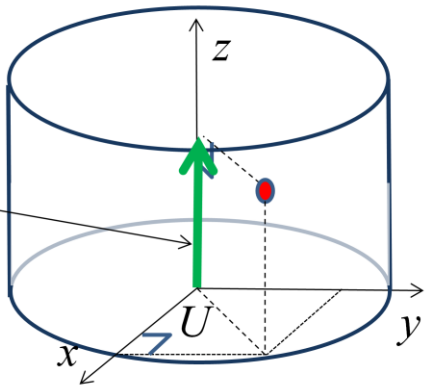
To get the normal vector of a parametric surface, we exploit isoparametric lines. Suppose that we need the normal at point associated with parameters u*, v*. Let us keep u* fixed, but allow v to run over its domain. This r(u*,v) is a one-variate parametric function, which is a curve. As it always satisfies the surface equation, this curve is on the surface and when v=v*, this curve passes through the point of interest. We know that the derivative of a curve always tangent to the curve, so the derivative with respect to v at v* will be the tangent of a curve at this point, and consequently will be in the tangent plane.

Similarly, the derivative with respect to u will always be in the tangent plane. The cross product results in a vector that is perpendicular to both operands, so it will be the normal of the tangent plane.

Cylinder

$x(U,V) = r\cos(U)$
$y(U,V) = r\sin(U)$
$z(U,V) = V$
$U \in [0,2\pi], \quad V \in [0,h]$

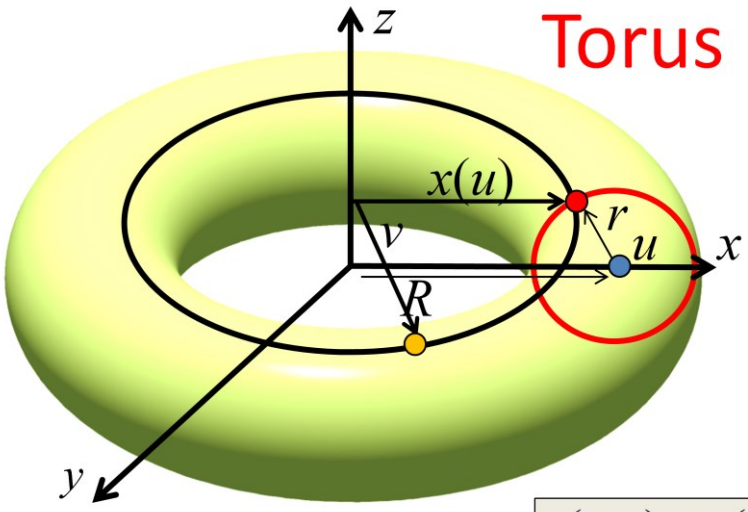
$\boldsymbol{s}(V) = (0,0,V)$



$\boldsymbol{b}(U) = (r\cos(U), r\sin(U), 0)$

```
void eval(float u, float v, vec3& point, vec3& normal) {
    float U = u * 2 * M_PI, V = v * height;
    vec3 base(cos(U) * r, sin(U) * r, 0), spine(0, 0, V);
    point = base + spine;
    normal = base;
}
```

Torus



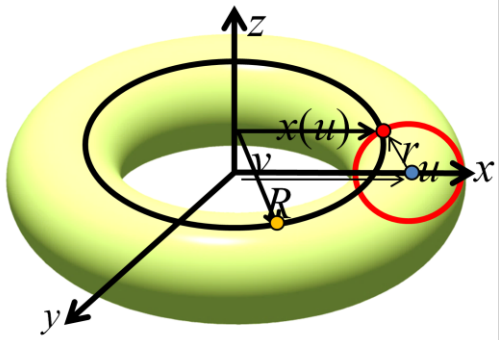
$$x(u) = R + r\cos(u)$$
$$z(u) = r\sin(u)$$

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

Torus

$$x(U,V) = \big(R + r\cos(U)\big)\cos(V)$$
$$y(U,V) = \big(R + r\cos(U)\big)\sin(V)$$
$$z(U,V) = r\sin(U)$$
$$U \in [0,2\pi],\quad V \in [0,2\pi]$$

D

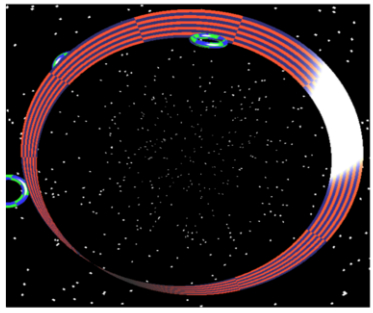



```
typedef Dnum<vec2> Dnum2;

void eval(float u, float v, vec3& point, vec3& normal){
   Dnum2 U(u*2*M_PI, vec2(1,0)), V(v*2*M_PI, vec2(0,1));
   Dnum2 D = Cos(U) * r + R;
   Dnum2 X = D * Cos(V), Y = D * Sin(V), Z = Sin(U) * r;
   point = vec3(X.f, Y.f, Z.f);
   vec3 drdU(X.d.x,Y.d.x,Z.d.x), drdV(X.d.y,Y.d.y,Z.d.y);
   normal = cross(drdU, drdV);
}
```

Möbius strip

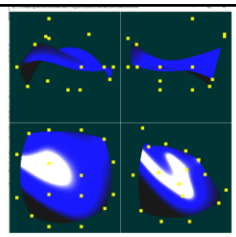
$$x(U,V) = \big(R + V\cos(U)\big)\cos(2U)$$
$$y(U,V) = \big(R + V\cos(U)\big)\sin(2U)$$
$$z(U,V) = V\sin(U)$$
$$U \in [0,\pi],\ \ V \in [-w/2, w/2]$$



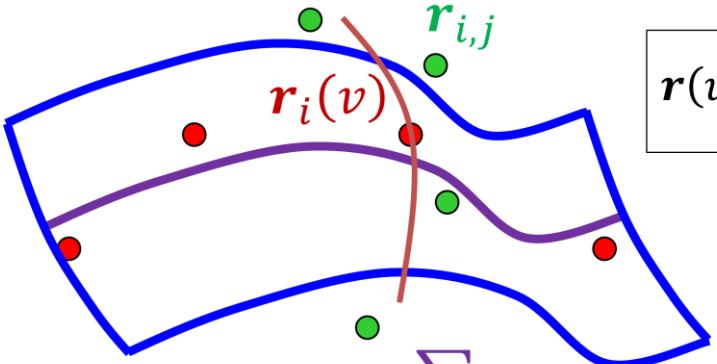
```
typedef Dnum<vec2> Dnum2;

void eval(float u, float v, vec3& point, vec3& normal){
   Dnum2 U(u*M_PI, vec2(1,0)), V((v-0.5)*w, vec2(0,1));
   Dnum2 X = (Cos(U) * V + R) * Cos(U * 2);
   Dnum2 Y = (Cos(U) * V + R) * Sin(U * 2);
   Dnum2 Z = Sin(U) * V;
   point = vec3(X.f, Y.f, Z.f);
   vec3 drdU(X.d.x,Y.d.x,Z.d.x), drdV(X.d.y,Y.d.y,Z.d.y);
   normal = cross(drdU, drdV);
}
```

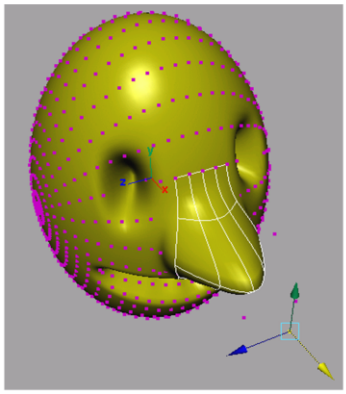
Free form surfaces



Definition with control points:



$$r(u,v) = \sum_i \sum_j B_j(v)\, B_i(u) r_{i,j}$$



$$r(u,v) = r_v(u) = \sum_i B_i(u)\, r_i(v)$$

$$r_i(v) = \sum_j B_j(v)\, r_{i,j}$$

The definition of curves traced back the problem to the specification of a few control points. We use the same approach here.

First, we trace back the definition of surfaces to curves. Let us fix one of the free variables of the surface, which results in a one-variable parametric form, a curve. This curve is on the surface and is called **isoparametric curve**. A curve can be well defined by control points. Now let us fix the isoparametric value differently, which leads to another isoparametric curve that can be defined with different control points. As the isoparametric value changes, the control points of the corresponding isoparametric curve also change. These changes are also curves, so we can express the path of the control point by blending other control points.

Substituting this into the equation of the isoparametric curve, we obtain the equation of the surface, which is a combination of control points forming a **control cage** or **control polyhedron**. The blending or weighting function of control point rij is the product of basis functions Bi parameterized with u, and Bj parameterized with v.