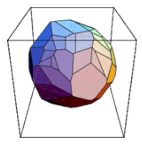
14. 3D Object Representation

> 💡 **3D Object Representation**
> - **Polyhedron**
> - **Quadric Surfaces**
> - **Blobby Object**
> - **Splines**
> - **Sweep Representation**
> - **Constructive Solid Geometry**

- CG 의 3대 분야
  - Modeling: 수학(기하) 기반 → 오늘 배울 부분
  - Rendering: 물리학(광학) 기반 → 여태껏 한 부분
  - Animation

Polyhedron

- **Polyhedron(다면체)** 점/선/면 → 다각형 (2D) → 다면체 (3D)
  - 다각형의 집합을 사용해서 다면체를 구성
  - 점(vertex), 선(edge), 면(face/facet) 으로 구성



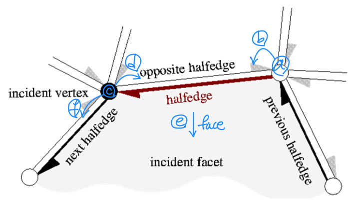
- **Data structure**

> 💡 Polyhedron 을 효과적으로 저장하기 위한 자료구조 사용
> (이유) 만약 다각형을 아무렇게나 저장한다면 쿼리들(ex. edge 와 인접한 vertex/face 찾기, vertex 로부터 출발하는 모든 edge 찾기 등) 을 빠르게 처리하는 게 불가능함

Half edge or doubly-connected edge list (DCEL)

- 대표적인 자료구조
- **edge 에 방향성을 두어** 양방향으로 가는 **2개의 half edge 로 정의**

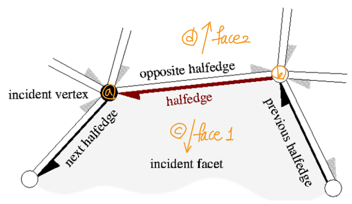
```
struct HE_vert {        // 꼭짓점의 데이터 구조
    float x; float y; float z; // x, y, z 좌표
    HE_edge* edge;    // 이 꼭짓점에서 출발하는 half-edge 중 하나를 point
};

struct HE_edge {        // half-edge의 데이터 구조
    HE_vert* vert;    // half-edge의 도착 vertex
    HE_edge* pair;    // 반대 방향 half-edge
    HE_face* face;    // half-edge가 둘러싸고 있는 face
    HE_edge* next;    // 같은 face를 둘러싸는 half-edge 중, 현재 half-edge의 다음 half-edge
};

struct HE_face {        // face의 데이터 구조
    HE_edge* edge;    // 면을 둘러싸는 half-edge 중 하나
};
```



Adjacency Queries using Half Edge

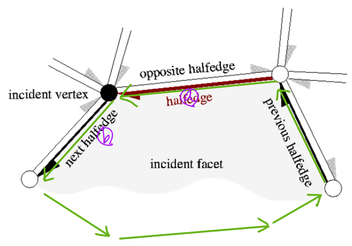
- (예시 1) edge 가 주어졌을 때, 인접한 vertex, face 를 찾기
  - 도착 vertex, 시작 vertex
  - 인접한 face 2개 (edge 는 보통 인접한 face 가 2개)

```
HE_vert* vert_e = edge->vert;             // 도착 vertex
HE_vert* vert_b = edge->pair->vert;    // 시작 vertex (pair의 도착 vertex)
HE_face* face1 = edge->face;           // edge에 인접한 첫 번째 face
HE_face* face2 = edge->pair->face;     // edge의 pair에 인접한 두 번째 face
```



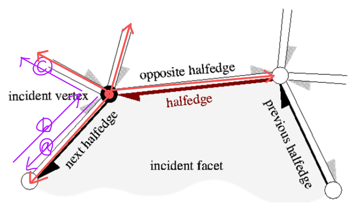
- (예시 2) face 가 주어져 있을 때, face 를 둘러싼 모든 half edge 들 찾기

```
HE_edge* edge = face->edge;      // face에 속한 하나의 half-edge를 가져옴
do {                             // edge의 next가 원래 자리로 돌아올 때까지 반복
        edge = edge->next;
} while (edge != face->edge);
```



- (예시 3) vertex 로부터 출발하는 모든 edge 들 찾기

```
HE_edge* edge = vert->edge;     // vertex에서 출발하는 한 edge를 가져옴
do {                            // 반대 방향 edge의 next가 원래 자리로 돌아올 때까지 반복
        edge = edge->pair->next;
} while (edge != vert->edge);
```

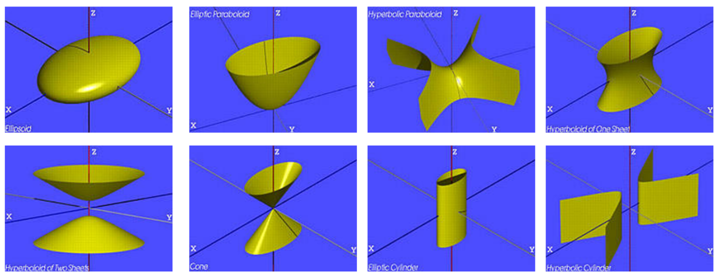


Quadric Surfaces

- **곡면**을 표현하는 방법 중 하나
- 계수를 어떻게 고르는 지에 따라 여러가지 형태의 곡면이 나옴

> 💡 (문제점) 내가 원하는 shape를 만들기 위해 계수들을 정해야 하는데, shape와 계수가 직관적으로 연결되지 않음
> → 어떤 식으로 바뀔지 사용자 입장에서 불분명 → 사용성 떨어짐

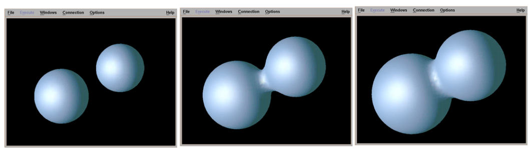
$$\sum_{i,j=1}^{D} Q_{i,j}x_i x_j + \sum_{i=1}^{D} P_i x_i + R = 0 \qquad ax^2 + by^2 + cz^2 + 2fyz + 2gzx + 2hxy + 2px + 2qy + 2rz + d = 0.$$

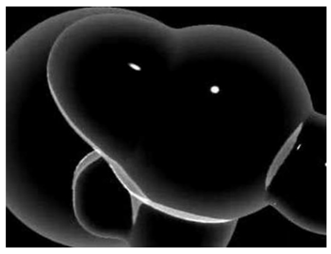


Blobby Object

- **곡면**을 표현하는 방법 중 하나
- **Non-rigid objects** 를 모델링
  - 고무, 분자, 액체, 물방울
  - 떨어져 있을 때는 독립적인 구 형태이다가, 가까워지면 연결되며 하나로 붙음

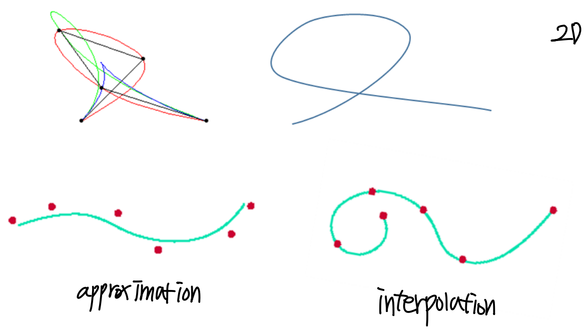
$$f(x, y, z) = \sum_k b_k e^{-a_k r_k^2} - T = 0 \qquad r_k^2 = x_k^2 + y_k^2 + z_k^2$$





Splines

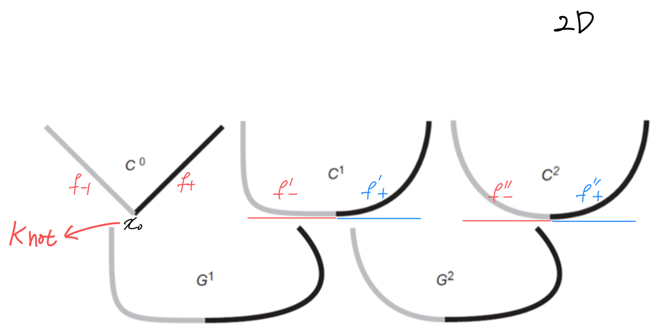
- **정의**
  - 일반적인 곡선, 곡면을 모델링할 때 사용
  - **Control points** 로 정의
  - Spline curve 의 차수 = Control points 의 개수 - 1
    - ex. 3개의 Control points 를 사용하는 경우, 차수는 2차
- **Approximation** vs **Interpolation**
  - Approximation: Spline curve 가 **모든 Control Points 를 통과하지는 않고, 근사**
  - Interpolation: Spline curve 가 **모든 Control Points를 정확히 통과**



- **Piecewise construction**
  - 하나의 복잡한 curve 를 만들기 위해, **차수가 낮은(대부분 3 이하) 여러 curve 를 연결하는 방식**

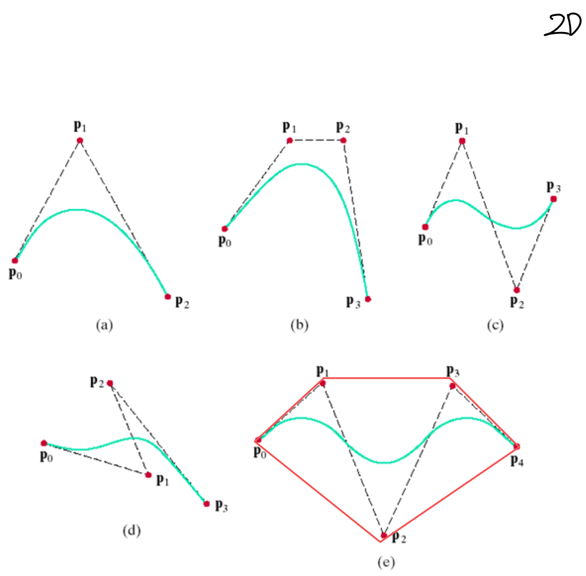
> 💡 사용하는 이유
> **⇒ 높은 차수 다항식의 문제점:** 높은 차수의 다항식은 곡선을 더 복잡하게 모델링할 수 있지만,
> 계산 중 오차가 누적될 가능성이 높음

> 💡 두 curve 가 **자연스럽게 연결되어야** 하나의 curve 같은 효과를 줄 수 있음
> → **Continuity** 가 중요



  - **Continuity (C^n vs G^n)**
    - C^n : 왼쪽 curve 와 오른쪽 curve 각각 n번 미분한 값이 **동일**
      - C^0: curve 의 끝점들이 정확히 일치
      - C^1: curve 를 1번 미분한 값(접선)이 동일 → Tangent continuous
      - C^2: curve 를 2번 미분한 값(곡률)이 동일 → Curvature continuous
      - n 이 클수록 smoothe 하게 연결
    - G^n : 왼쪽 curve 와 오른쪽 curve 각각 n-1번 미분한 값이 **동일(C^n-1)** + n번 미분한 값이 **배수 관계** → 방향성 같다
  ⇒ 보통 C^2, G^2 사용
- Bezier, b-spline, NURBS
  - 모두 다항 함수이지만, 표현하기 위한 Basis function 에 따라 종류가 달라짐

Bezier Curves and Surfaces



- **Bezier curve**
  - Spline curve 의 일종으로, **Bernstein Polynomials** 을 Basis function 으로 사용

$$\mathbf{p}(u) = \sum_{k=0}^{n} \mathbf{p}_k b_{k,n}(u), \qquad 0 \le u \le 1$$

  - n: Curve 의 차수, p_k: Control point, **$b_{k,n}(u)$**: n차 **Bernstein Polynomials, P(u): Bezier curve**
    - n차 Bezier curve 는 n+1개의 Control point 로 정의
    - **P(u)**는 (x(u), y(u)) → 2차원 Bezier curve 위의 점
- **Bernstein Polynomials 특성**

$$b_{k,n}(u) = {}_nC_k u^k (1-u)^{n-k}, \qquad {}_nC_k = \frac{n!}{k!\,(n-k)!}, 0 \le k \le n$$

0차 $\{b_{0,0}(u) = 1\}$

1차 $\begin{Bmatrix} b_{0,1}(u) = 1-u \\ b_{1,1}(u) = u \end{Bmatrix}$

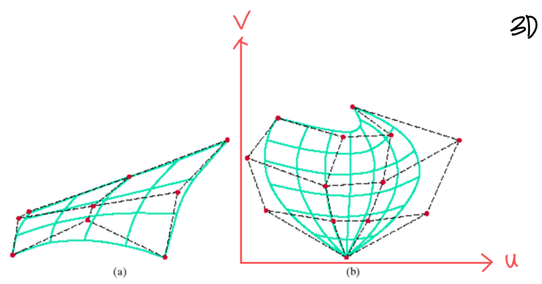
$$\begin{Bmatrix} b_{0,2}(u) = (1-u)^2 \\ b_{1,2}(u) = 2(1-u)u \\ b_{2,2}(u) = u^2 \end{Bmatrix} \qquad \begin{Bmatrix} b_{0,3}(u) = (1-u)^3 \\ b_{1,3}(u) = 3(1-u)^2 u \\ b_{2,3}(u) = 3(1-u)u^2 \\ b_{3,3}(u) = u^3 \end{Bmatrix}$$

$$\textstyle\sum_{k=0}^{n} b_{k,n}(u) = 1$$

  - 모든 u 에 대해 **$b_{k,n}(u)$** 은 항상 양수 (→ Curve 가 control point 의 가중 평균 형태를 가지게 함)
  - 합이 1 (→ Curve 의 전체 합이 일정하게 유지)

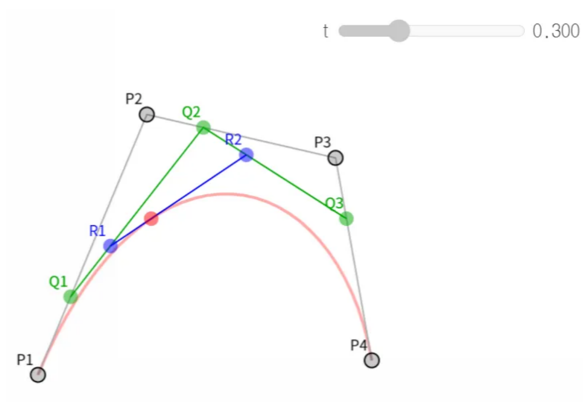
> 💡 ⇒ 이 두 특성으로 인해, **Bezier curve 는 Convex hull propety 를 가짐**

> 💡 **Convex hull property**
> - Convex hull: Control points 를 모두 포함하는 최소한의 볼록 다각형
> - Bezier curve P(u) 는 절대로 Convex hull 바깥으로 나가지 않음
>
>   → Bezier curve 는 Convex hull 안에 포함되므로, Bounding volume 으로 사용 가능



- Surface의 경우
  - u → **(u, v)** 로 확장 (0 ≤ u ≤ 1, 0 ≤ v ≤ 1)

De Casteljau's algorithm



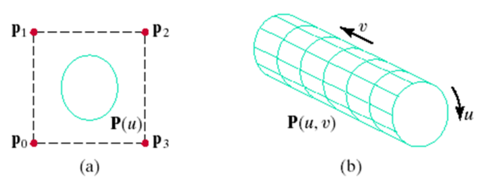
- Bezier curve 를 그리는 기하학적 방법
- P(u) 를 다항식으로 직접 계산하지 않고, **Recursive 한 Linear interpolation 을 사용**
- 방법
  - Recursive 하게 매개변수 t 에 따라 **(1-t) : t 비율**로 내분점을 찾음 (0 ≤ t ≤ 1)
  - t=0 이면 Curve 의 시작점 P1, t=1 이면 끝점 P4

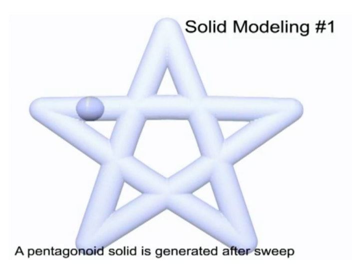
> 💡 t=0.3 일 때, Control points 개수 4개(P1, P2, P3, P4) → 3차 Bezier curve 를 그려보자

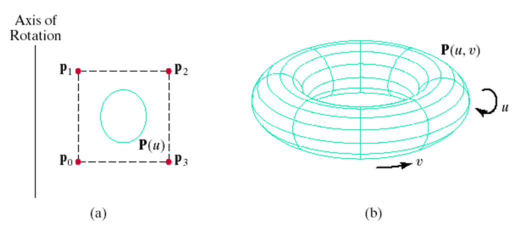
1. **P1-P2, P2-P3, P3-P4** 1:3 내분점 찾기 → **Q1, Q2, Q3**
2. **Q1-Q2, Q2-Q3** 1:3 내분점 찾기 → **R1, R2**
3. **R1-R2** 1:3 내분점 찾기 → Curve 상의 한 점

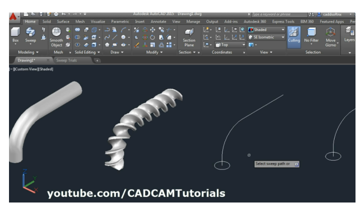
Sweep Representation

1. Profile curve (간단한 2D 형태) 를 정의
2. Profile curve 를 어느 방향으로 Sweep 할 것인지 지정
   - 원을 직선 방향으로 sweep → 실린더
   - 원을 축을 기준으로 회전시키며 sweep → 도넛 모양









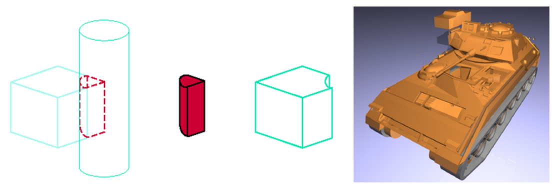
Constructive Solid Geometry

- 복잡한 Shape 를 만들기 위해, 간단한 Shape 들에 **Boolean operation(Union, Intersection, Difference)** 을 하는 방법

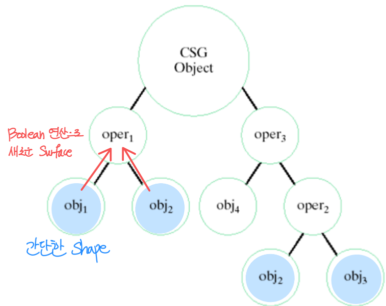
  합집합 교집합 차집합

> 💡 실제로는 매우 많은 Boolean operation 이 필요 → CSG tree 사용

- **CSG tree**
  - Leaf node: 간단한 Shape
  - Internal Node: Boolean operation
- 동작 방식:
  - CSG tree 의 Leaf node 에 있는 간단한 Shape 들을 Boolean operation 을 통해 조합하여 최종 결과물을 생성



8456 CSG operations



Quiz) de Castlejau's Algorithm

Given the following cubic Bezier curve P(u) in 2D in Eq1, we want to find the point $\mathbf{P}(\frac{1}{2})$ on the curve:

Eq1: $\mathbf{P}(u) = \sum_{k=0}^{3} \mathbf{P}_k b_k(u)$, where $b_k(u) = \frac{3!}{k!(3-k)!} u^k (1-u)^{(3-k)}$

with control points $\mathbf{P}_0 = (-8,4), \mathbf{P}_1 = (-4,4), \mathbf{P}_2 = (0,0), \mathbf{P}_3 = (4,-4)$.

- Fill the blanks in the following with a decimal number, not a fractional number (e.g. 0.5 instead of 1/2 or $\frac{1}{2}$).
- If you click <체크>, your answer will be verified and scored immediately.

문제 1
정답
총 4.00 점에서 4.00 점 할당
문제 표시

Find $\mathbf{P}(\frac{1}{2})$ =(-2 ✔ , 1.5 ✔) by directly evaluating $u = \frac{1}{2}$ in Eq1.

$\mathbf{P}(\frac{1}{2}) = \sum_{k=0}^{3} \mathbf{P}_k \frac{3!}{k!(3-k)!} (\frac{1}{2})^k (1-\frac{1}{2})^{(3-k)}$

$\mathbf{P}(\frac{1}{2})$ =(-2,1.5).

$P(\frac{1}{2}) = \sum_{k=0}^{3} P_k \cdot \frac{3!}{k!(3-k)!} \times (\frac{1}{2})^k (1-\frac{1}{2})^{3-k}$

$= P_0 \times {}_3C_0 \times (\frac{1}{2})^0 (\frac{1}{2})^3 \rightarrow (-8,4) \times \frac{1}{2^3}$

$+ P_1 \times {}_3C_1 \times (\frac{1}{2})^1 (\frac{1}{2})^2 \rightarrow (-4,4) \times 3 \times \frac{1}{2^3}$

$+ P_2 \times {}_3C_2 \times (\frac{1}{2})^2 (\frac{1}{2})^1 \rightarrow (0,0) \times 3 \times \frac{1}{2^3}$

$+ P_3 \times {}_3C_3 \times (\frac{1}{2})^3 (\frac{1}{2})^0 \rightarrow (4,-4) \times \frac{1}{2^3}$

$= \frac{1}{2^3} \{ (-8,4) + (-12,12) + (0,0) + (4,-4) \}$

$= \frac{1}{2^3} (-16, 12) = (-2, \frac{3}{2})$

정보
문제 표시

We want to evaluate $\mathbf{P}(\frac{1}{2})$ by using de Casteljau's algorithm.

문제 2
정답
총 6.00 점에서 6.00 점 할당
문제 표시

The first three points generated by de Casteljau's algorithm using $\overline{\mathbf{P}_0\mathbf{P}_1}, \overline{\mathbf{P}_1\mathbf{P}_2}, \overline{\mathbf{P}_2\mathbf{P}_3}$ are $\mathbf{P}'_0$ = (-6 ✔ , 4 ✔), $\mathbf{P}'_1$ =(-2 ✔ , 2 ✔), $\mathbf{P}'_2$ =(2 ✔ , -2 ✔).

$\mathbf{P}'_0 = \frac{1}{2}(\mathbf{P}_0 + \mathbf{P}_1)$
$\mathbf{P}'_1 = \frac{1}{2}(\mathbf{P}_1 + \mathbf{P}_2)$
$\mathbf{P}'_2 = \frac{1}{2}(\mathbf{P}_2 + \mathbf{P}_3)$
$\mathbf{P}'_0$ = (-6,4), $\mathbf{P}'_1$ =(-2,2), $\mathbf{P}'_2$ =(2,-2).

$P(\frac{1}{2})$ 이면 1:1 내분점.

P_0 $(-8,4)$ P_1 $(-4,4)$ P_2 $(0,0)$ P_3 $(4,-4)$

$\left(\frac{(-8\times1)+(-4\times1)}{2}, \frac{(4\times1)+(4\times1)}{2}\right) = (-6,4)$

$\left(\frac{(-4\times1)+(0\times1)}{2}, \frac{(4\times1)+(0\times1)}{2}\right) = (-2,2)$

$\left(\frac{(0\times1)+(4\times1)}{2}, \frac{(0\times1)+(-4\times1)}{2}\right) = (2,-2)$

$\left(\frac{(-6\times1)+(-2\times1)}{2}, \frac{(4\times1)+(2\times1)}{2}\right) = (-4,3)$

$\left(\frac{(-2\times1)+(2\times1)}{2}, \frac{(2\times1)+(-2\times1)}{2}\right) = (0,0)$

$\left(\frac{(-4\times1)+(0\times1)}{2}, \frac{(3\times1)+(0\times1)}{2}\right) = (-2,\frac{3}{2})$

문제 3
정답
총 4.00 점에서 4.00 점 할당
문제 표시

The second two points generated by de Casteljau's algorithm using $\overline{\mathbf{P}'_0\mathbf{P}'_1}, \overline{\mathbf{P}'_1\mathbf{P}'_2}$ are $\mathbf{P}''_0$ = (-4 ✔ , 3 ✔), $\mathbf{P}''_1$ =(0 ✔ , 0 ✔).

$\mathbf{P}''_0 = \frac{1}{2}(\mathbf{P}'_0 + \mathbf{P}'_1)$
$\mathbf{P}''_1 = \frac{1}{2}(\mathbf{P}'_1 + \mathbf{P}'_2)$
$\mathbf{P}''_0$ = (-4,3), $\mathbf{P}''_1$ =(0,0).

문제 4
정답
총 2.00 점에서 2.00 점 할당
문제 표시

The final point generated by de Casteljau's algorithm using $\overline{\mathbf{P}''_0\mathbf{P}''_1}$ is $\mathbf{P}(\frac{1}{2})$ =(-2 ✔ , 1.5 ✔).

$\mathbf{P}(\frac{1}{2}) = \frac{1}{2}(\mathbf{P}''_0 + \mathbf{P}''_1)$.
$\mathbf{P}(\frac{1}{2})$ =(-2,1.5).