Computer Graphics

Prof. Jibum Kim

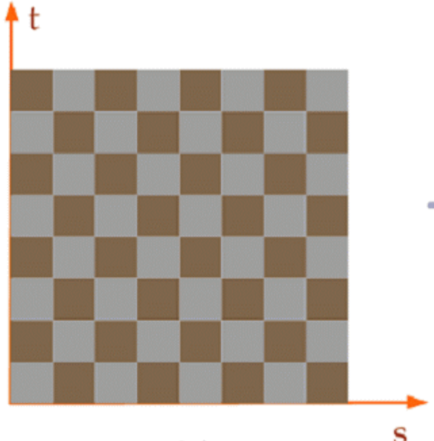
Department of Computer Science & Engineering

Incheon National University

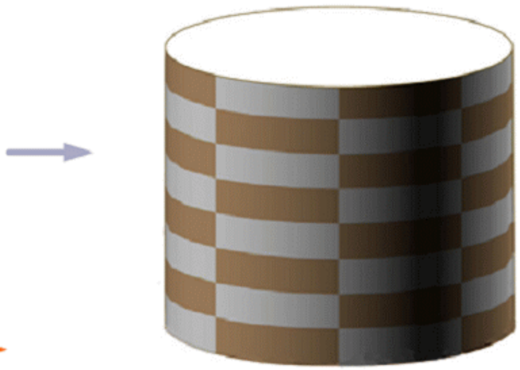
- **Parametric surfaces**

- Our programs so far have been simple from the point of view of specifying the texture map
- The surfaces textured were all polygons, so that all we had to was specify texture coordinates at the corners
- How about more complicated surfaces?
- It is actually surprisingly straightforward if the surface is parametried

- 곡면에 texture mapping을 수행하는 일반적인 방법은 곡면의 parametric form을 이용하는 것이다. 아래와 같이 평면에 있는 2차원의 Texture를 매개 변수를 이용하여 곡면에 texture mapping을 하는 방법에 대하여 공부해 보자
- 먼저, 곡면을 매개 변수 (u ,v)로 표현할 수 있는 경우에 대해서 살펴보자
- 이를 곡면의 parametric form (매개변수)을 이용한 표현이라고 한다
- 이러한 경우 비교적 손쉽게 texture space와 곡면을 mapping 시킬 수 있다

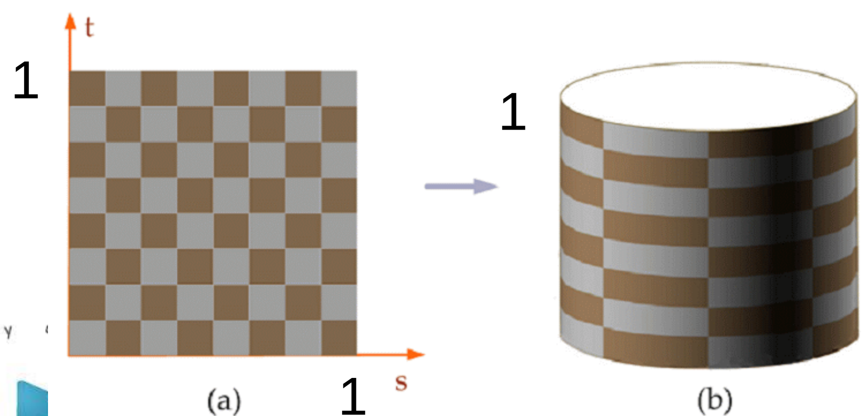


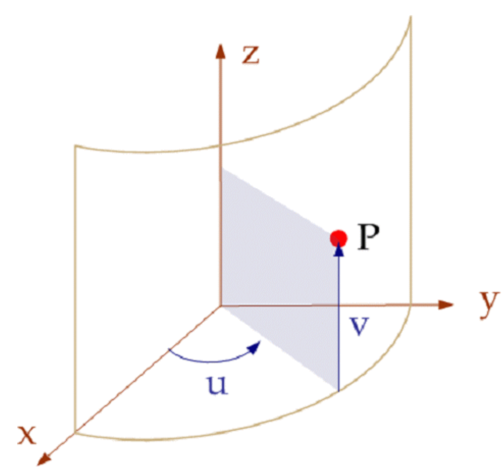
(a)


(b)

- Cylindrical coordinate (원통 좌표계), Cartesian coordinate (직각 좌표계)
- 원기둥 표면 상의 점 P (x, y, z)는 parametric form으로 P (r, u, v) 로 표현 가능
- r은 원기둥의 반지름으로 고정된 값이면 P(u, v) ↔ P(x, y, z)로 mapping
- u: 점 P를 x-y평면에 투영시 양의 x축 방향에서 반 시계 방향으로 측정한 각 u
- v: 점 P의 양의 z축에 대한 높이

$$x = r\cos u, \; y = r\sin u, \; z = v$$

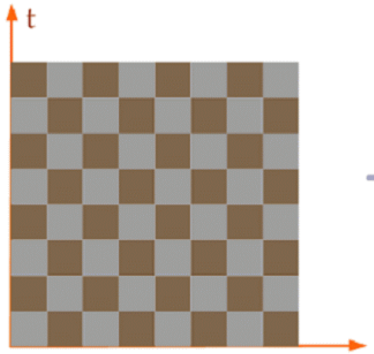




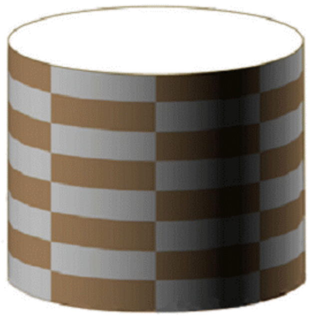
- Texture space에서 s: [0, 1] , t: [0, 1] 에서 정의됨. u: [0, 2π], v: [0, 1] 정의
- 즉, s가 0에서 1로 갈 때, u는 0에서 2π 각도를 커버해야 한다
- t가 0에서 1로 갈 때, v는 0에서 1로 가면 된다
- $u = 2\pi s, v = t$, 앞에서 $x = r\cos u, \; y = r\sin u, \; z = v$

$$\boxed{x = r\cos 2\pi s, \; y = r\sin 2\pi s, \; z = t}$$

- 즉, Texture space 상의 (s, t)를 원통 표면의 (x, y, z)로 mapping 가능

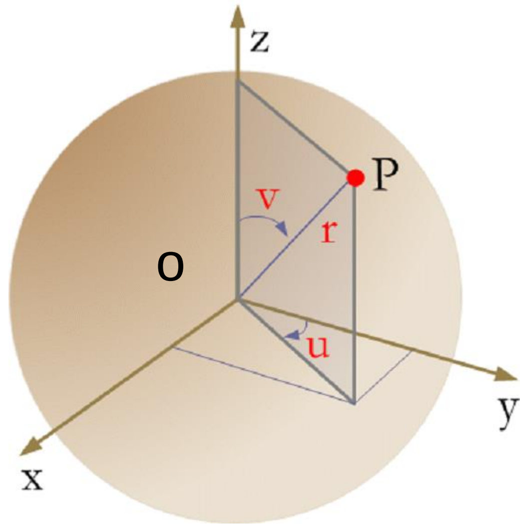


(a)


(b)

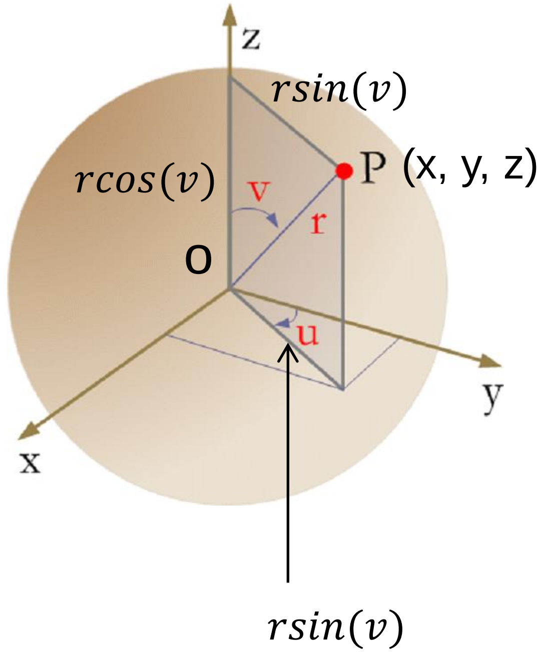
- 이와 같은 곡면의 parametric form을 이용한 texture mapping은 비슷한 방법으로 구 (sphere)에도 적용 가능하다
- 지구를 구라고 생각하면 지구 표면 위의 어떤 위치를 위도와 경도로 나타내는 것과 유사하다
- 구 표면 상의 점 P (x, y, z)는 P (r, u, v) 로 표현 가능
- r은 원기둥의 반지름으로 고정된 값이면
- P(u, v) ↔ P(x, y, z)로 mapping

- v: 선분 OP와 z축의 양의 방향이 이루는 각
- u: P를 x-y 평면에 투영했을 y축의양의 방향과 이루는 각



- **Spherical coordinates, Cartesian coordinates**
- 구 표면 상의 점 P (x, y, z)는 P (u, v) 로 표현 가능
- r: 구의 반지름 (고정 값)
- v : 선분 OP와 z축의 양의 방향이 이루는 각
- u: P를 x-y 평면에 투영했을 y축의
- 양의 방향과 이루는 각

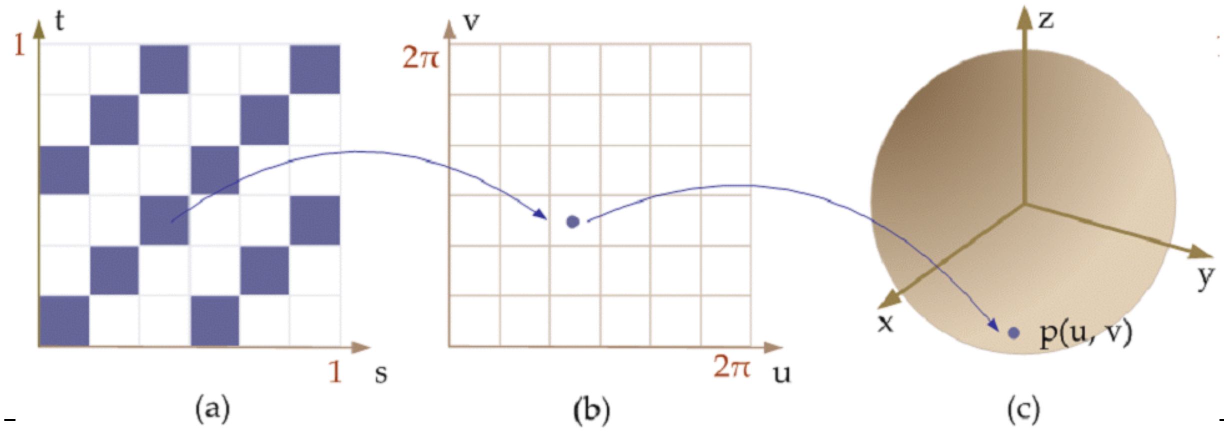
$$\begin{aligned} z &= r\cos v \\ y &= r\sin v\ \cos u \\ x &= r\sin v\sin u \end{aligned}$$



인천대학교

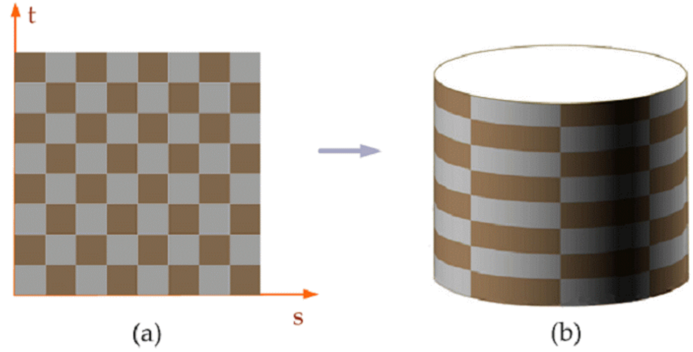
- Texture space에서 s: [0, 1] , t: [0, 1] 에서 정의됨
- u: [0, 2π], v: [0, 2π]에서 정의
- 즉, s가 0에서 1로 갈 때 u는 0에서 2π 각도를 커버
- t가 0에서 1로 갈 때 v는 0에서 2π 각도를 커버
- $u = 2\pi s$, $v = 2\pi t$, 이 식들을 앞에 식에 대입

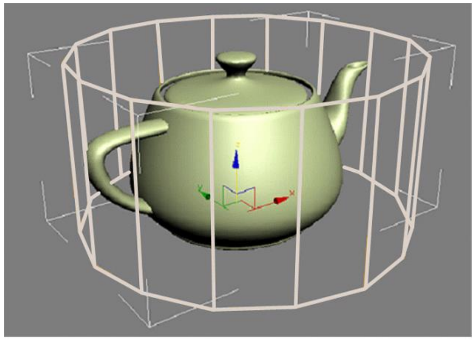
$$
\boxed{\begin{aligned}
z &= r\cos 2\pi t \\
y &= r\sin 2\pi t\,\cos 2\pi s \\
x &= r\sin 2\pi t\,\sin 2\pi s
\end{aligned}}
$$



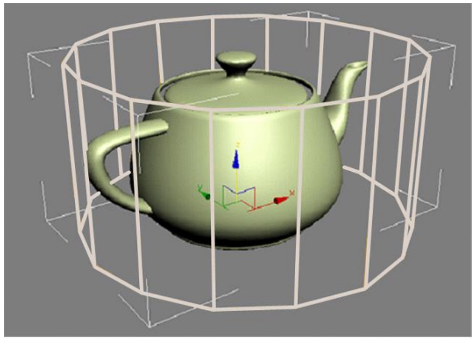
- **2-stage mapping을 이용한 곡면의 texture mapping**

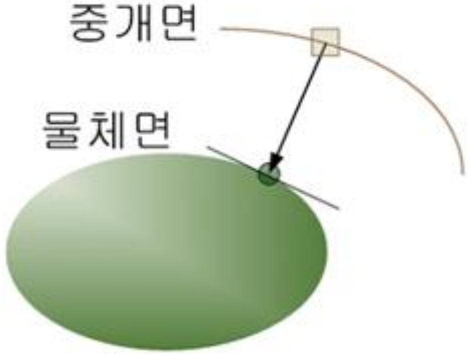
- 많은 경우에 곡면이 원기둥이나, 구가 아니다. 이러한 경우에는 어떻게 texture mapping을 진행해야 하나?
- 이러한 경우에는 2단계 mapping (2-stage mapping)을 이용한다
- 1단계 mapping: Texture를 중개면 (intermediate surface)에 입힌다 (이를 S-mapping이라 부른다)
- 중개면으로는 앞에서 배운 구와 cylinder 등이 흔히 사용된다



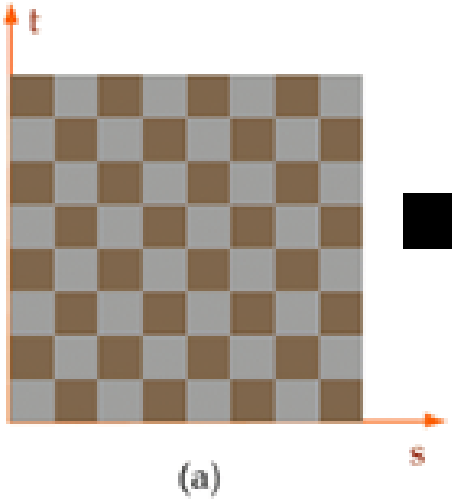


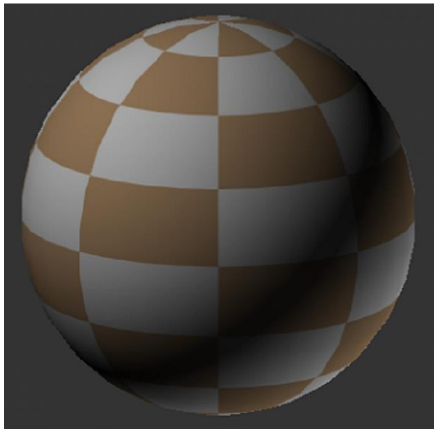
- **2단계 mapping: O mapping**
- 물체를 중개면 내부에 넣고 물체면에 texture를 입히는 작업이다
- 중개면으로는 구, cylinder등을 사용한다
- 여러가지 O mapping 방법이 있는데 한가자 방법은 물체면의 법선 벡터가 중개면과 만나는 점 (위치)를 구한 뒤, 그 점의 texture값을 해당 물체면의 texture로 사상한다



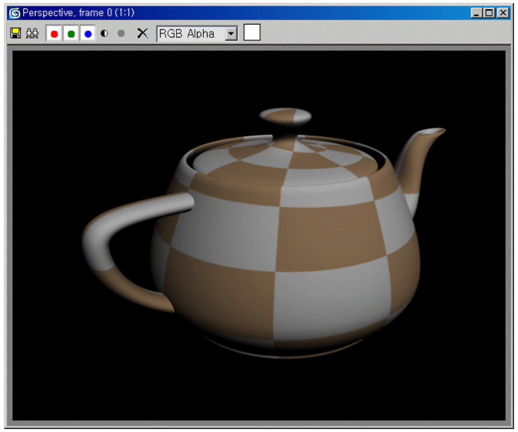


- **주전자 (곡면)에 대한 2단계 mapping (2-stage mapping)의 예**





1단계 mapping



2단계 mapping

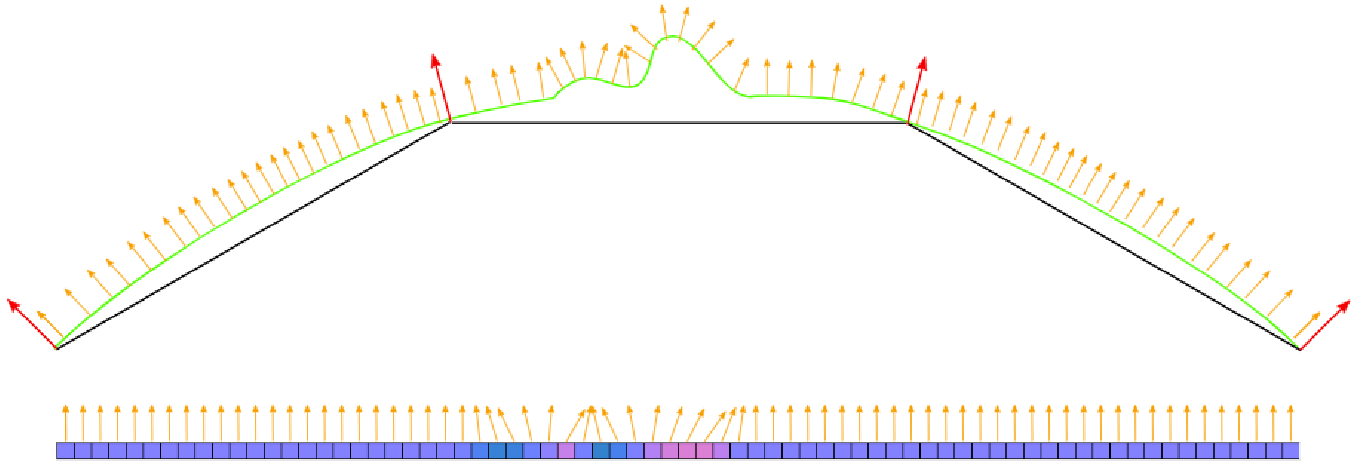
- **Parametric form을 이용한 torus로의 texture mapping 예**
- **https://www.dropbox.com/s/r8sos7h02urxjty/texture_2.txt?dl=0**

인천대학교
University of Incheon

- **Bump mapping**

- **Bump mapping**
- It is a highly effective method for **adding complex texture** to objects without having to alter their underlying geometry
- For example, the below figure shows a close-up view of an orange with a rough pitted surface
- However, the actual geometry of the orange is a smooth sphere, and the texture has been painted on it
- At each point, the texture **perturbs (교란?) the natual normal vector to the surface** and consequently perturbs the normal direction that is so important in the calculation of each specular highlight

- **Unity bump mapping**
- [Unity - Manual: Normal map (Bump mapping) (unity3d.com)]



Normal mapping across three polygons, viewed as a 2D diagram

- Blinn developed a method called bump mapping to give the illusion of geometric detail on a surface by means of perturbing the surface normal, but without actually changing any geometry
- The idea is to re-align the normal to the original surface so that light reflects from it as if it were detailed
- A one-dimensional example will make matters clear

- **Bump mapping**

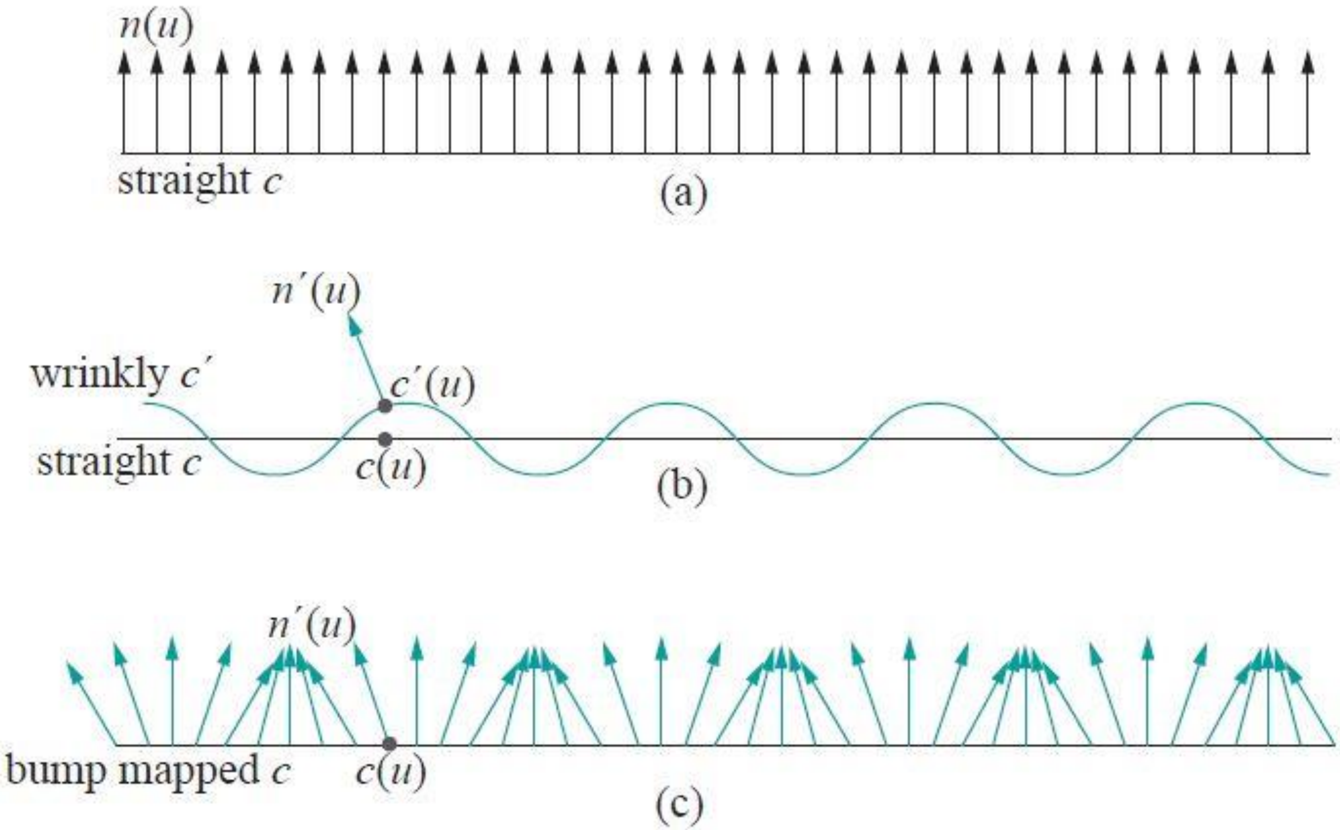
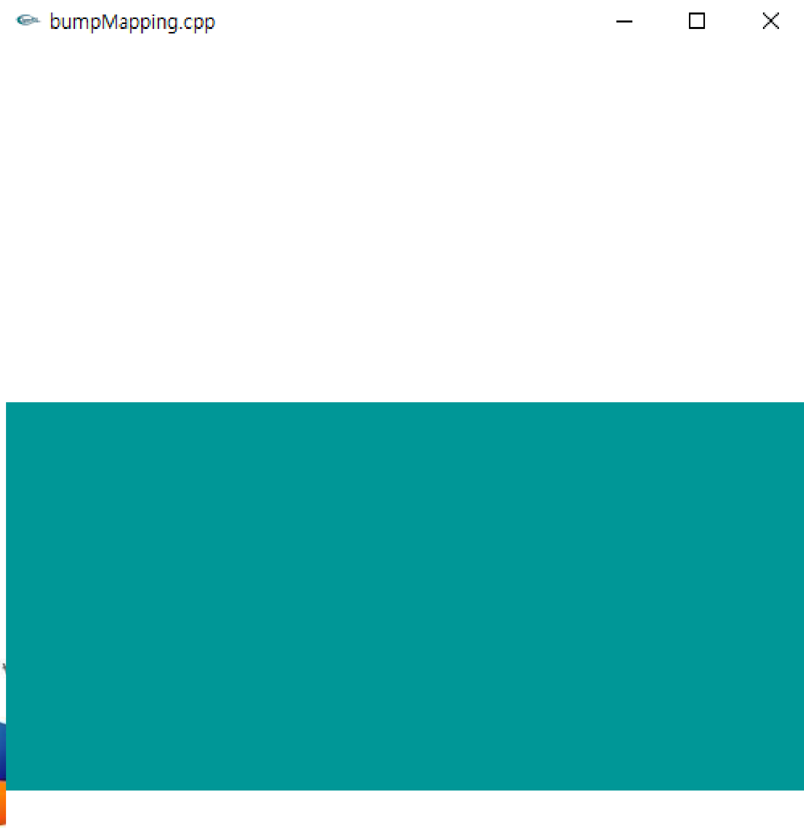
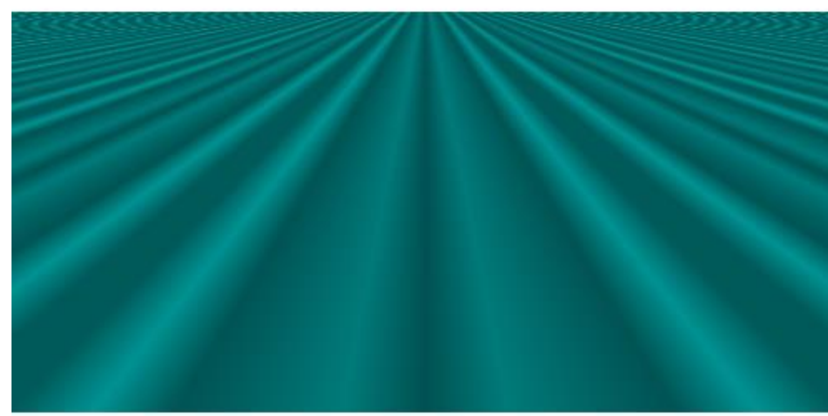


Figure 13.31: Bump mapping: (a) The original curve c and its true unit normals $n(u)$ (b) The wrinkled curve c' and its unit normal $n'(u)$ at a single point $c'(u)$ (c) Bump mapped c with redefined normals $n'(u)$.

- **Chapter 13/BumpMapping**

- 다음은 plane에 bump mapping을 적용 전 후의 비교이다
- ‘space’ 키로 bump mapping 전, 후를 비교해 보자





- **Curves and surfaces**

- **Until now we have worked with flat entities such as lines and flat polygons**
  - Fit well with graphics hardware
  - Mathematically simple

- **But the world is not composed of flat entities**
  - Need curves and curved surfaces

- 1D objects are unions of straight and curved segments
- Parts composed of straight segments can be drawn exactly in an OpenGL environement – one would invoke the GL_LINES, GL_LINE_STRIP and GL_LINE_LOOP primitives.
- Curved segments, on the other hand, have to be approximated

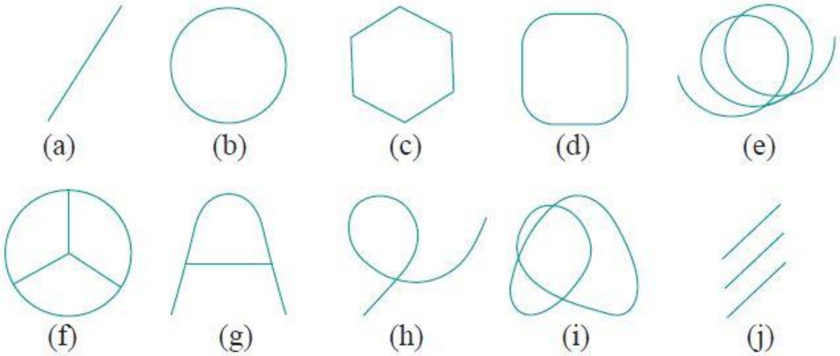


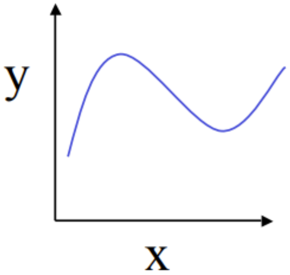
Figure 10.1: One-dimensional objects.

- **Three major types of object representation**
- **Explicit**
- **Implicit**
- **Parametric**
- **Consider the advantages and disadvantages of each form**

- **Explicit representation**
- **The explicit form of a curve in 2D gives the value of one variable, the dependent variable (종속 변수), in terms of the other, the independent variable (독립 변수).**
- **In x, y space, we might write**

  $y = f(x)$ **or** $x = g(y)$

- **There is no guarantee that either form exists for a given curve**
- **For the line, we write the equation as** $y = mx + h$



- The problem of explicit representation is it cannot represent all curves
- 예: circles
- It write one equation for half of it
- $y = \sqrt{r^2 - x^2}$ and a secand equation,
- $y = -\sqrt{r^2 - x^2}$ for the other half
- In addition, we must also specify that these equations hold only if $0 \leq |x| \leq r$

- Implicit representation
- Most of the curves and surfaces with which we work have implicit representations
- In 2D, an implicit curve can be represented by the equation

$$f(x, y) = 0$$

Line: ax+by+c=0

Circles: $x^2 + y^2 - r^2 = 0$

인천대학교

- In 3D, the implicit form
- $f(x, y, z) = 0$ describes a surface
- Plane: $ax + by + cz + d = 0$
- Sphere: $x^2 + y^2 + z^2 - r^2 = 0$
- An implicit equation gives a Boolean condition for points on the curve to satisfy: a point lies on the curve $f(x, y) = 0$; it doesn't if $f(x, y) \neq 0$
- However, it gives us no analytic way to find a value u on the curve that corresponds to a given x

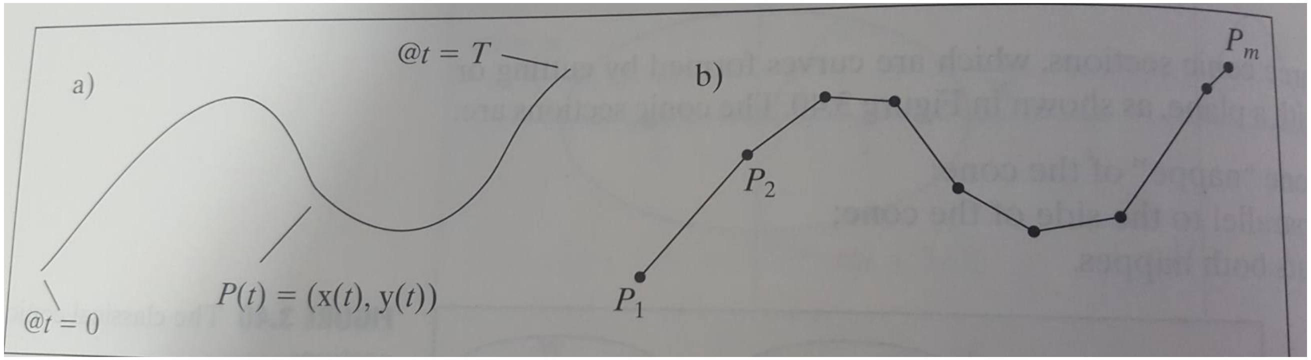
- Parametric form
- The parametric form of a curve expresses the value of each spatial variable for points on the curve in terms of an independent variable, t, the parameter.
- Here are parametrizations of the curves
- Line
- Parametric: $x = t, y = -\frac{a}{b}t - \frac{c}{b}, t \in (-\infty, \infty)$
- Implicit: $ax + by + c = 0$
- Drawing a curve from its parametric equations is straightforward. This is a major advantage of the parametric for

인천대학교

- **Circle**
- **Parametric:** $\boldsymbol{x = rcos(t), y = rsin(t), t} \in [-\infty, \infty]$
- **Implicit:** $\boldsymbol{x^2 + y^2 = r^2}$

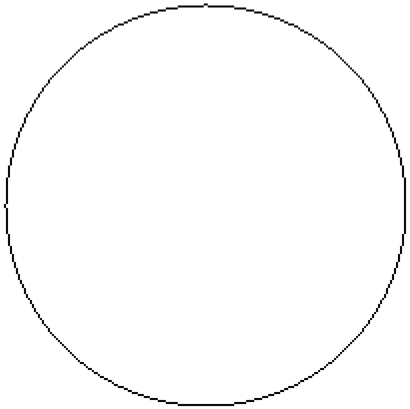
- **Drawing curves**

- Suppose a curve C has the parametric representation $P(t) = \big(x(t), y(t)\big)$ as t varies from 0 to T as shown below
- Take samples of $P(t)$ at closely spaced instants
- The curve $P(t)$ is then approximated by the polyline

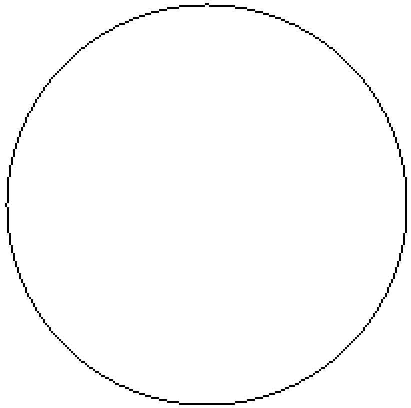


- $x^2 + y^2 = 1$ 인 원을 앞의 방법으로 그려보자
- $x = cos(t), y = sin(t), 0 \leq t \leq 2\pi,$ N=100

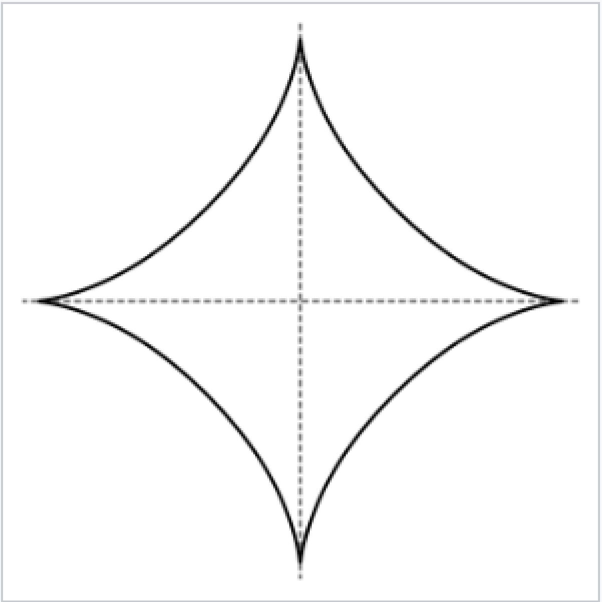
- #define TWOPI 2*3.141592
- double t=0; // 각도
- int N=100;   // 100개의 sample 사용
- glBegin(GL_LINE_STRIP);
- for(t=0; t<= 2*PI; t+=2*PI/N)
- {
- glVertex2f(cos(t), sin(t));
- }
- glEnd();



- 예: LINE_STRIP을 사용하여 원을 근사화한 예
- https://www.dropbox.com/s/snvp0ldza01m4dg/circle.txt?dl=0



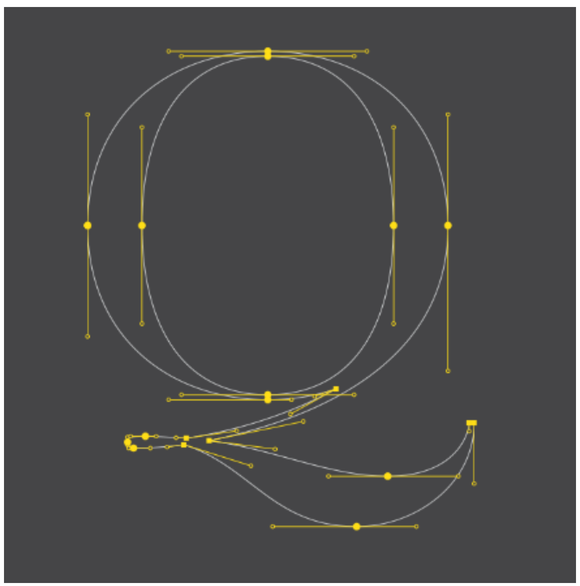
- **Astroid**
- **Implicit:** $x^{2/3} + y^{2/3} - 1 = 0$
- **Parametric:** $x = cos^3 t,\ y = sin^3 t,\ 0 \le t \le 2\pi$
- **Chapter 10/astroid.cpp**



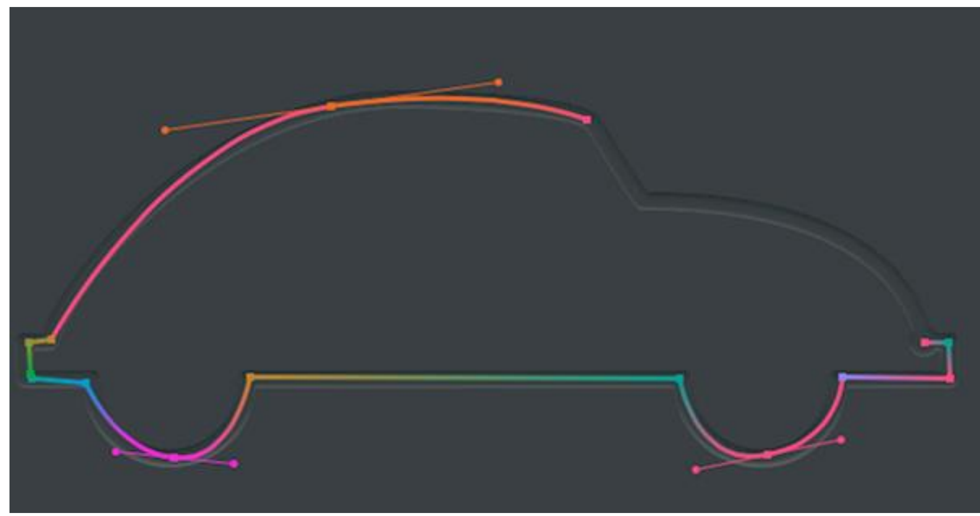
Astroid

- **Interactive한 곡선 및 곡면 설계**

- **자동차, 비행기의 외형 , 로고 등을 설계할 때에도 곡선, 곡면이 필요하다**

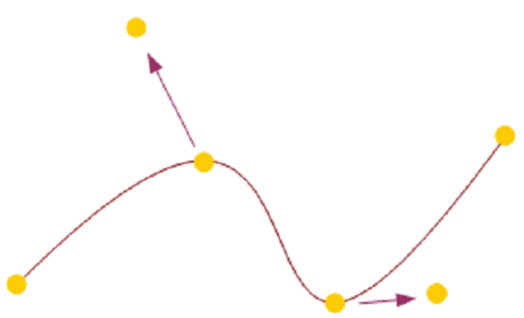


curve 를 이용한 Logo 디자인

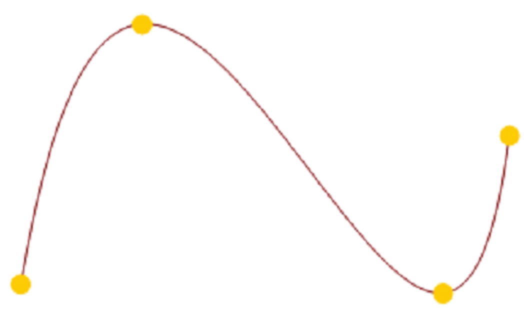


curve 를 이용한 차 디자인

- 곡선 (curve)
- 곡선의 모양을 제어 및 결정하는 특징적인 점들을 **control points (제어점)** 이라고 한다
- 사용자는 이러한 제어점을 추가, 삭제, 위치 변경들을 통해 곡선의 모습을 제어할 수 있다
- 예; https://www.youtube.com/watch?v=GC0OK8j7-B8

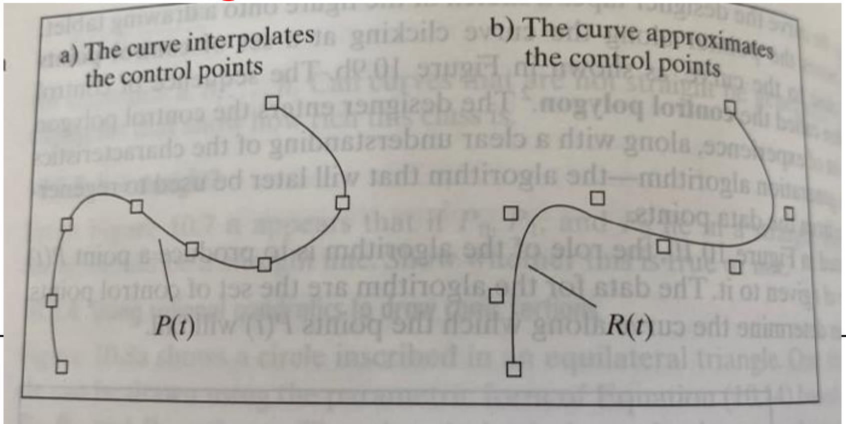


(a)

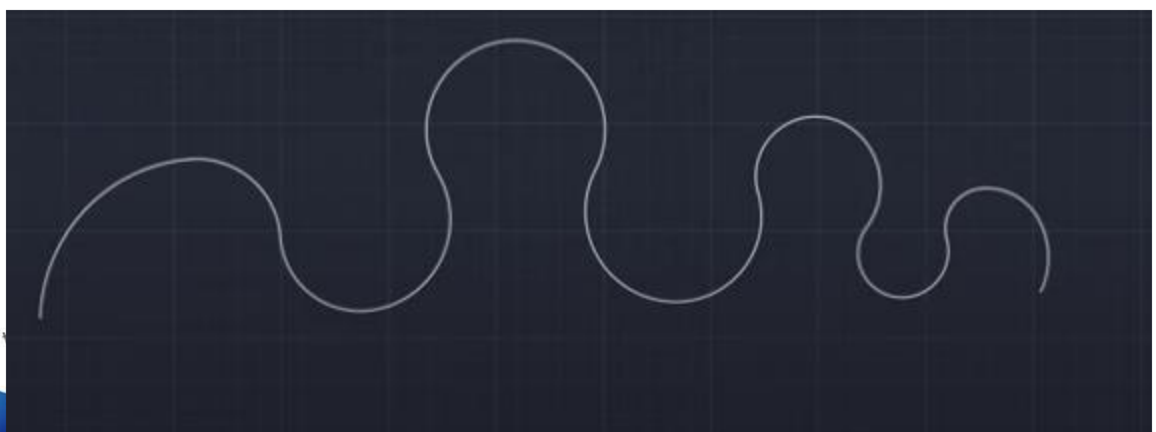


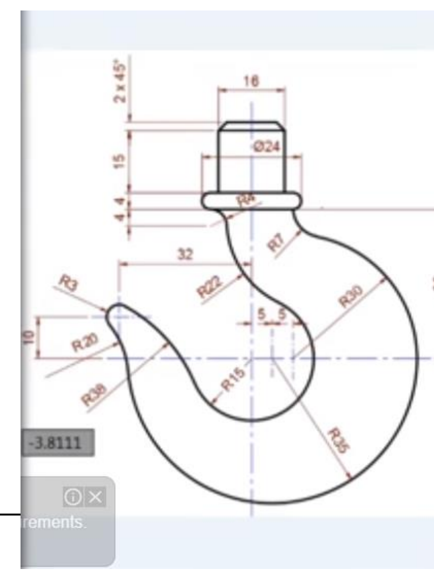
(b)

- **Interpolating curve (보간 커브)**
- **(a) shows an algorithm that generates a curve P(t) that interpolates the control points and forms a smooth curve for points in between**
- **Approximating curve (근사 커브)**
- **(b) uses an algorithm that generates a curve R(t) that approximates the control points**
- **R(t) is attracted toward each control point in turn, but doesn’t actually pass through all of them**

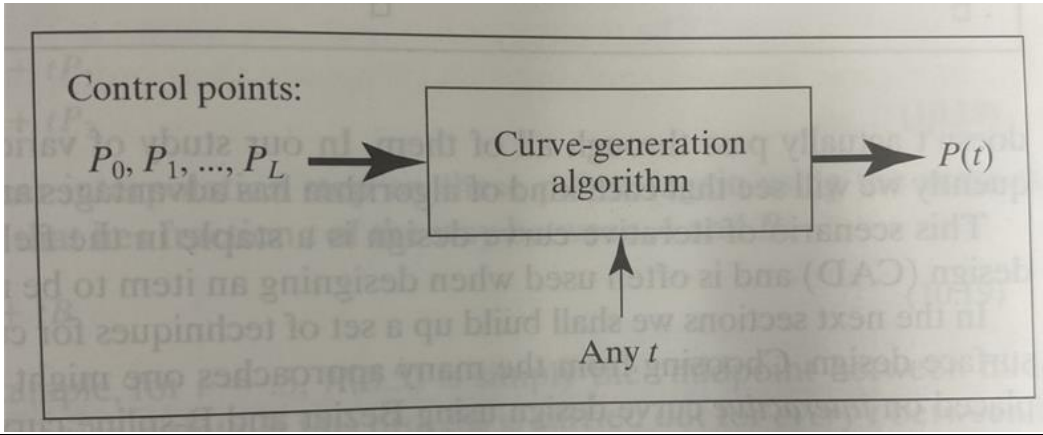


- In our study of various algorithms, we will see that each kind of algorithm has advantages and disadvantages
- This scenario of iterative curve design is a staple in the field of computer-aided design (CAD) and is often used when designing an item to be manufactured

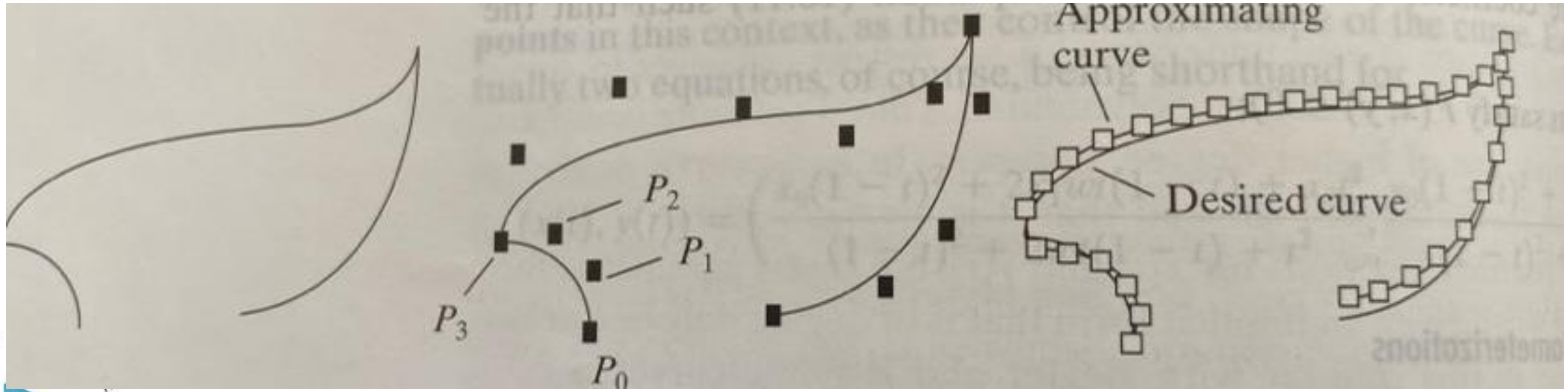




- Interactive design process
- 1. Lay down the initial control points
- 2. Use the algorithm to generate the curve
- 3. If the curve is satisfactory, stop
- 4. Adjust some control points
- 5. Go to step 2



- **A curve design scenario**
- **왼쪽: desired curve**
- **가운데: control points**
- **오른쪽: 알고리즘과 approximating curve로 만들어진 curve 결과**



- $B\acute{e}zier$ **curve (베지에 곡선)**

- ***Bézier* (베지에) curve**
- **Suppose a programmer specifies P_0, P_1,…,P_n of n+1 control points, asking for a curve not necessarily passing through them, but whose shape is molded by the control points**
- **The generated curve is said to approximate the control points**
- **Below figure is a curve approximating five control points**
- **Bezier and de Casteljau independently invented a particular method to approximate a sequence of control points**

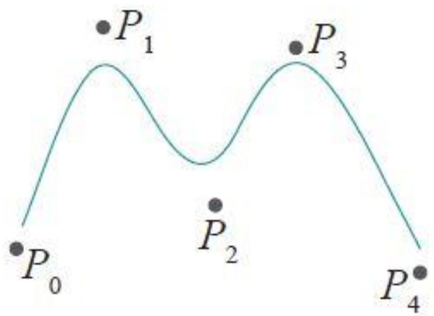
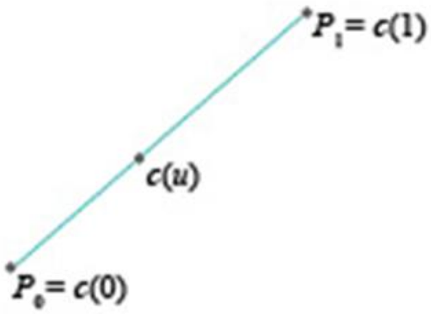


Figure 18.1: A curve approximating five control points.

- **Linear (선형) Bezier curves**
- **Let's start with the simplest case, where there are only two control points P0 and P1**
- **The Bezier curve c approximating P0 and P1 is simply the straight line segment joining two**
- **We write the parametric equation of c as follows**
- $\boldsymbol{c(u) = (1-u)P_0 + uP_1}$ **, 단,** $\boldsymbol{0 \leq u \leq 1}$
- **The Bezier curve is said to be linear order**
- **being the number of control points**



- 예: what is the equation of the linear Bezier curve c with control points $\begin{bmatrix} \mathbf{5} \\ \mathbf{1} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{-1} \\ \mathbf{0} \end{bmatrix}$?
- What are the points on c corresponding to the values 0, 0.3 and 1 of the parameter u?

- Observations
- 1. The parametric equation for c is linear in u, is why it is called a linear Bezier curve
- $c(u) = (1-u)P_0 + uP_1$, 단, $0 \le u \le 1$

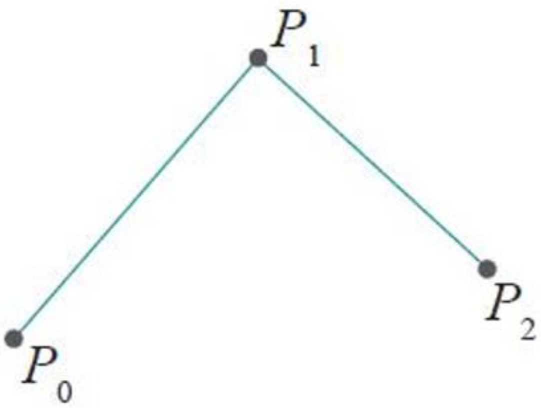
- 2. The curve c can be thought of as a weighted sum of P_0 and P_1. The weights are $1-u$ and u
- These functions are called the blending (basis) functions of the respective control points
- The blending functions 1-u and u are known as the Bernstein polynomials of degree 1, which is denoted as $B_{0,1}(u)$ and $B_{1,1}(u)$
- $c(u) = B_{0,1}(u)P_0 + B_{1,1}(u)\ P_1$, 단, $0 \le u \le 1$

- $c(u) = (1-u)P_0 + uP_1$, 단, $0 \le u \le 1$
- $c(u) = B_{0,1}(u)P_0 + B_{1,1}(u)\ P_1$, 단, $0 \le u \le 1$
- **The blending function $B_{0,1}(u)$ decreases from 1 to 0 as u goes from 0 to 1, while exactly the opposite is true of that of the second control point**
- **3. $B_{0,1}(u)$ and $B_{1,1}(u)$ both lie between 0 and 1**

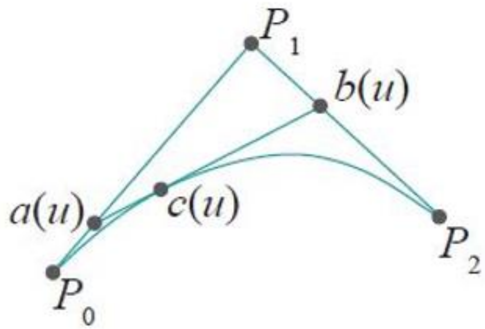
  $B_{0,1}(u) + B_{1,1}(u) = 1$ **for each u, 단, $0 \le u \le 1$**

- **Quadratic Bezier curve**

- Consider next three control points $\boldsymbol{P_0}$, $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$.
- We want to construct the Bezier curve approximating these control points
- One possible way is to linearly interpolate between $\boldsymbol{P_0}$ and $\boldsymbol{P_1}$ and then between $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$
- However the corner at $\boldsymbol{P_1}$ makes unsatisfactory



- Casteljau resolves the problem by adding a third interpolation step
- 1. First interpolate between P_0 and P_1 to find the point
- $\boldsymbol{a(u) = (1-u)P_0 + uP_1}$
- 2. Next interpolate between P_1 and P_2 to find the point
- $\boldsymbol{b(u) = (1-u)P_1 + uP_2}$
- 3. Finally interpolate between $a(u)$ and $\mathbf{b}(u)$ to determine the point
- $\boldsymbol{c(u) = (1-u)a(u) + ub(u)}$
- 대입하면 $\mathbf{c}(u) = (1-u)^2 P_0 + 2u(1-u)P_1 + u^2 P_2,\ 0 \le u \le 1$



- $\mathbf{c}(u) = (1-u)^2 P_0 + 2u(1-u)P_1 + u^2 P_2,\ 0 \le u \le 1$
- **This describes the quadratic Bezier curve approximating three control points P_0, P_1, and P_2, which is indeed smooth**

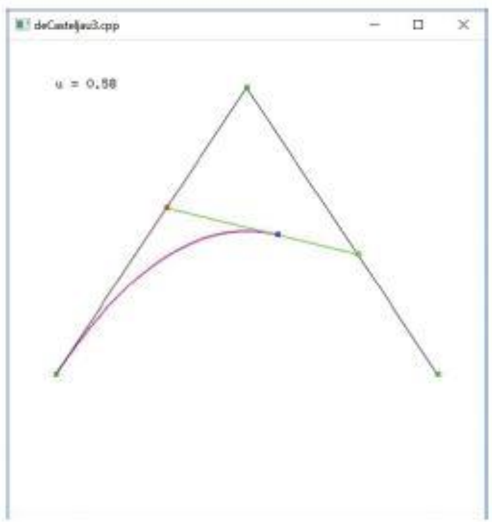


Figure 18.4: Screenshot of deCasteljau3.cpp.

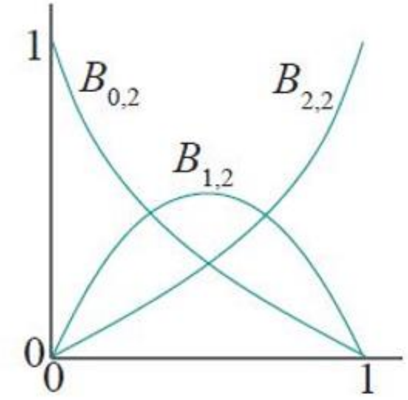
- **Chapter 18: DeCasterlu3.cpp**
- **Three control points**
- **Press the left or right arrow keys to decrease or increase the curve parameter u**

- 예: what is the equation of the quadratic Bezier curve c with control points $\begin{bmatrix} \mathbf{0} \\ \mathbf{-1} \end{bmatrix}, \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{5} \\ \mathbf{-1} \end{bmatrix}$?

- Observations
- 1. c is quadratic in u
- 2. $c(u) = (1-u)^2 P_0 + 2u(1-u)P_1 + u^2 P_2,\ 0 \le u \le 1$
- Blending function은 Berstein polynomials of degree 2
- $c(u) = B_{0,2}(u)P_0 + B_{1,2}(u)\ P_1 + B_{2,2}(u)\ P_2,$
- 단, $0 \le u \le 1$
- 3. $B_{0,2}(u), B_{1,2}(u), B_{2,2}(u)$ both lie between 0 and 1

  $B_{0,2}(u) + B_{1,2}(u) + B_{2,2}(u) = 1$

  for each u, 단, $0 \le u \le 1$



- Bezier curve는 종종 행렬 형태로도 표현되는데
- $\mathbf{c}(u) = (1-u)^2 P_0 + 2u(1-u)P_1 + u^2 P_2,\ 0 \le u \le 1$
- 는
- $\mathbf{c}(u) = \begin{bmatrix} P_0 & P_1 & P_2 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} u^2 \\ u \\ 1 \end{bmatrix}$