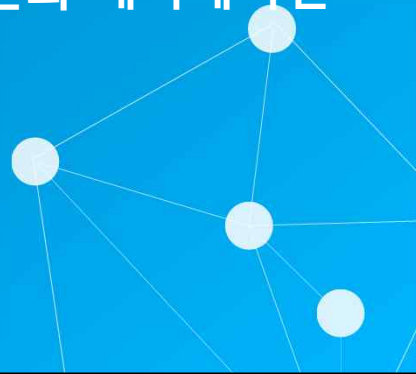


# 06

CHAPTER

## 시점 변환과 애니메이션



### 6장. 학습 내용



- 관측(Viewing)의 개요
- OpenGL의 시점 변환
- 응용: 시점 변환을 통한 애니메이션
- 로봇의 애니메이션: 달리는 로봇

## 6.1 관측(Viewing)의 개요



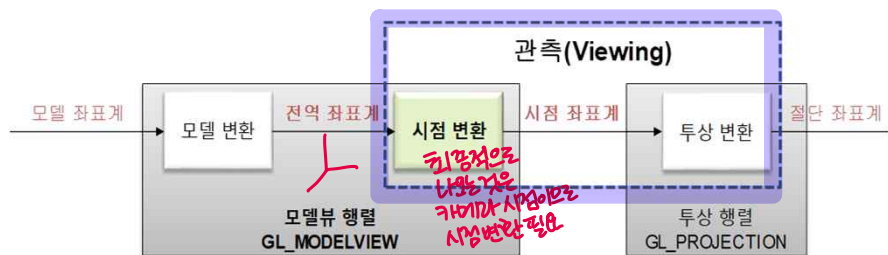
- 모델 변환
  - 객체들을 3차원 공간(전역 좌표)에 표현하는 방법
- 관측(Viewing)
  - 카메라를 다루는 과정
  - "시점 변환 + 투상 변환"

3

## 관측이란?



- 시점 변환(6장) + 투상 변환(7장)



4

## 6.2 OpenGL의 시점 변환



- 시점(viewpoint)과 시점 좌표계
- 시점 변환
  - 시점 좌표계를 전역 좌표계와 일치시키는 과정
- OpenGL의 시점 변환 함수
- 시점 변환 함수의 호출 위치

5

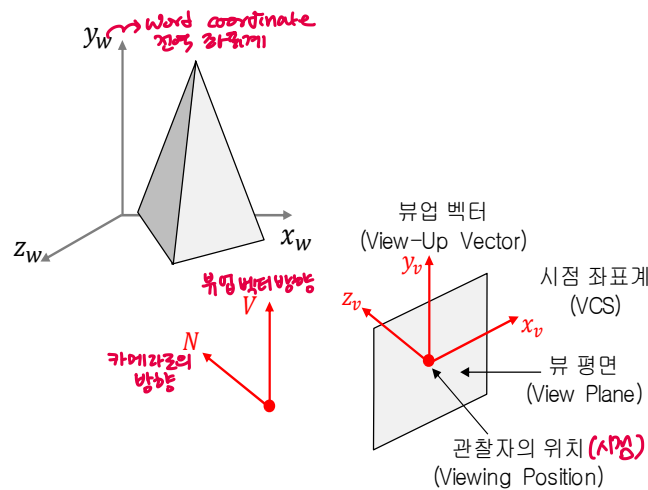
## 시점(viewpoint)과 시점 변환



- 시점(viewpoint):
    - 물체를 바라보는 위치, 또는 카메라의 위치
  - 시점 변환
    - 시점 만을 설정하는 것은 아님
    - 시점 뿐 아니라 카메라가 어느 곳을 보는지, 그리고 카메라의 기울기는 어떤지 등의 정보도 포함
    - 좌표계 변환의 일종
- ☆ **시점(Viewing) 좌표계(VCS)**를 설정하고 전역 좌표계로 표시된 객체의 정보를 시점 좌표계 기준으로 바꾸는 작업

6

## 시점 좌표계의 개념



7

## 시점(viewpoint)과 시점 변환

- 관찰자의 위치(View Reference Point, viewing position):
  - 카메라의 위치, 또는 눈의 위치
  - 시점 좌표계의 원점
- 뷰 평면의 법선 벡터(View-Plane Normal Vector)
  - 카메라가 향하는 방향
  - 시점 좌표계의 z축과 나란한 것으로 봄
  - 뷰 평면의 법선 벡터 N으로 정의
- 뷰업 벡터(View-Up Vector)
  - 벡터 N을 축으로 카메라를 회전 → 영상의 회전을 의미
  - Portrait / Landscape
  - 시점과 N, V까지 설정하면 최종적인 시점 좌표계 완성

8

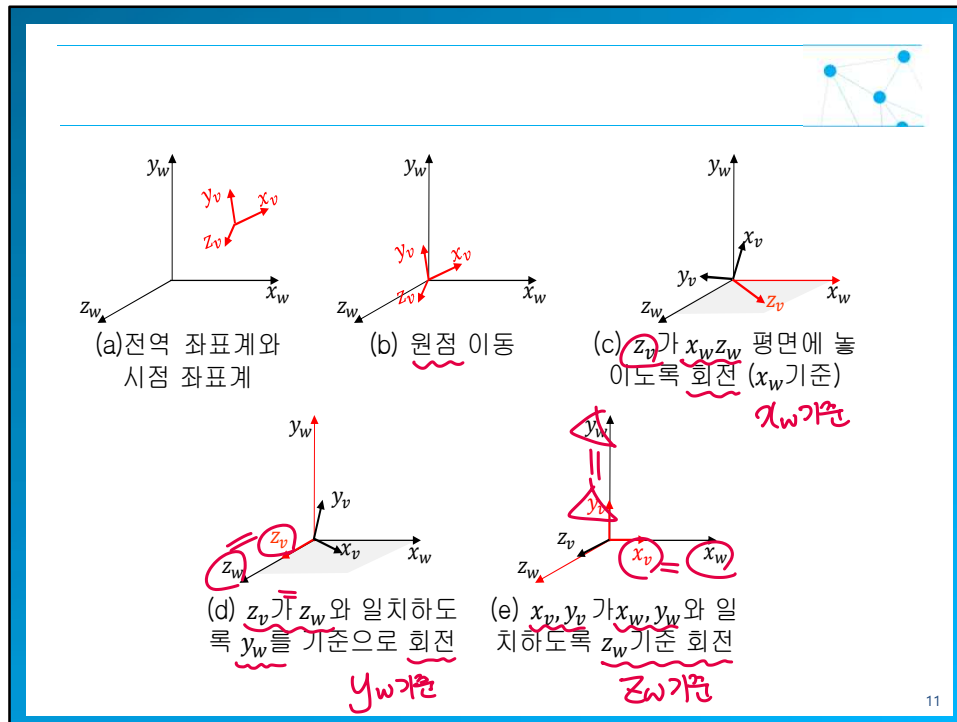
## 시점 변환: 시점 좌표계를 전역 좌표계와 일치시키는 과정

- 모델뷰 행렬을 행등행렬로 초기화
  - MCS, WCS 및 VCS가 모두 일치
- 모델 변환
  - 객체의 정점들은 MCS에서 WCS를 기준으로 변환
  - 여전히 WCS와 VCS와 동일  $WCS = VCS$
- 시점 변환
  - 드디어 VCS를 WCS에서 분리하는 작업
  - WCS상에 객체들이 있고, VCS를 변환하여 다양한 관측 가능
  - 예) 모델 변환을 통해 로봇을 합체한 후 시점 변환을 통해 로봇을 여러 방향에서 살필 수 있음
- 시점 변환도 좌표계 변환의 일종

## 시점 변환 과정

- (1) 시점 좌표계의 원점을 전역 좌표계의 원점으로 이동시킨다.
- (2) 회전 변환을 통해 좌표축  $x_v$ ,  $y_v$  그리고  $z_v$ 를 전역 좌표계  $x_w$ ,  $y_w$ ,  $z_w$ 와 일치시킨다. 이 과정은 생각보다 약간 복잡하다. 다음은 한 예를 설명한다.
  - 먼저 하나의 축, 예를 들어  $x_w$ 를 중심으로 좌표축을 회전하여  $z_v$ 가  $x_w z_w$  평면에 놓이도록 한다.
  - 다음으로  $y_w$ 를 중심으로 좌표축을 회전하여  $z_v$ 가  $z_w$ 와 일치하도록 한다.
  - 마지막으로  $z_w$ 를 중심으로 회전하여  $x_v$ ,  $y_v$ 가  $x_w$ ,  $y_w$ 와 각각 일치하도록 한다.

2번



11

## OpenGL의 시점 변환 함수

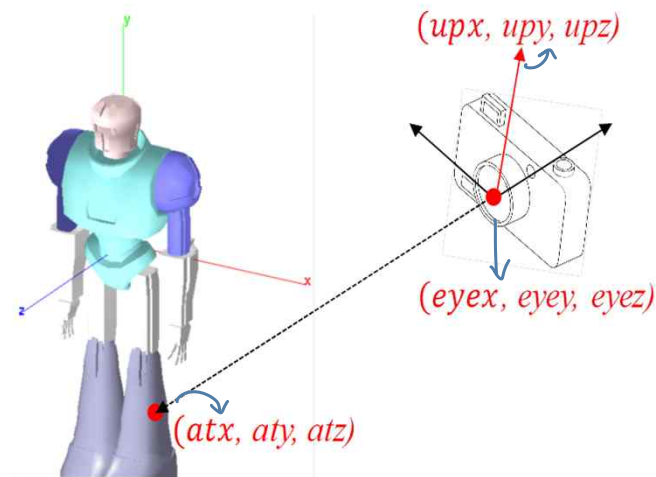
- 방법 1: 기본 변환 함수 사용
  - `glTranslate()`, `glScale()`, `glRotate()` 등
- 방법 2: GLU 함수 사용

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
               GLdouble atx, GLdouble aty, GLdouble atz,
               GLdouble upx, GLdouble upy, GLdouble upz);
```

- (eyex, eyey, eyez): 카메라 위치
- (atx, aty, atz): 카메라가 바라보는 점의 위치
- (upx, upy, upz): 뷰업 벡터. 카메라 기울임

12

## gluLookAt()함수의 파라미터 의미



13

## 시점 변환 함수의 호출 위치

- 시점 변환 함수 호출의 위치가 모델 변환을 위한 함수 호출 이전에 이루어져야 함

$$P_{WCS} = M \cdot P_{MCS}$$

$$P_{VCS} = V \cdot P_{WCS} = \underbrace{V}_{\text{시점}} \cdot \underbrace{M}_{\text{모델}} \cdot P_{MCS}$$

- 전형적인 방법

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();           // CTM <- I  
gluLookAt(0,0,0, 0,0,-10, 0,1,0); // 시점변환: CTM <- CTM · V = I · V  
glRotatef(45, 0,1,0);       // 모델변환: CTM <- I · V · M  
glutWireCube(1.0);         // 정점변환: P' <- I · V · M · P
```

포인터입장

함수 호출입장  
(모델변환 입장)

14

## 6.3 응용: 시점 변환을 통한 애니메이션

- Lab: 마우스를 이용한 로봇의 회전
- Lab: `glRotate()`을 이용한 로봇의 회전
- Lab: `gluLookAt()`을 이용한 로봇의 회전

15

## 마우스를 이용한 로봇의 회전

```
4. static Robot robot;
5. static float RotX = 0.0f; // X축 중심의 회전 각
6. static float RotY = 0.0f; // Y축 중심의 회전 각
7.
8. void display() {
9.     glClearColor(0.99, 0.97, 0.97, 1.0);
10.    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
11.
12.    // 시점 변환 코드
13.    glMatrixMode(GL_MODELVIEW);
14.    glLoadIdentity(); // MODELVIEW행렬 <==> 항등행렬
15.    glRotated(RotY, 0, 1, 0); // 시점변환: Y축 중심의 회전
16.    glRotated(RotX, 1, 0, 0); // 시점변환: X축 중심의 회전
17.
18.    // 모델 변환 코드
19.    robot.draw();
20.
21.    glutSwapBuffers();
22.    glFlush();
23. }
```

16



```

24. void keyboard(unsigned char key, int x, int y) {
25.     if (key == 'i') {
26.         glMatrixMode(GL_MODELVIEW);
27.         glLoadIdentity();
28.         RotX = RotY = 0.0f;
29.     }
30.     // 'w', 's', 'q' 콜백 처리 및 이후 코드는 프로그램 5.3과 동일
31. }
32. static int PrevX, PrevY;
33. void mouseClick(int button, int state, int x, int y) {
34.     if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
35.         PrevX = x;
36.         PrevY = y;
37.     }
38. }
39. void mouseMotion(GLint x, GLint y) {
40.     RotX += (PrevY-y);
41.     RotY += (PrevX-x);
42.     PrevX = x;
43.     PrevY = y;
44.     glutPostRedisplay();
45. }

```

17

## 시점변환을 이용한 애니메이션

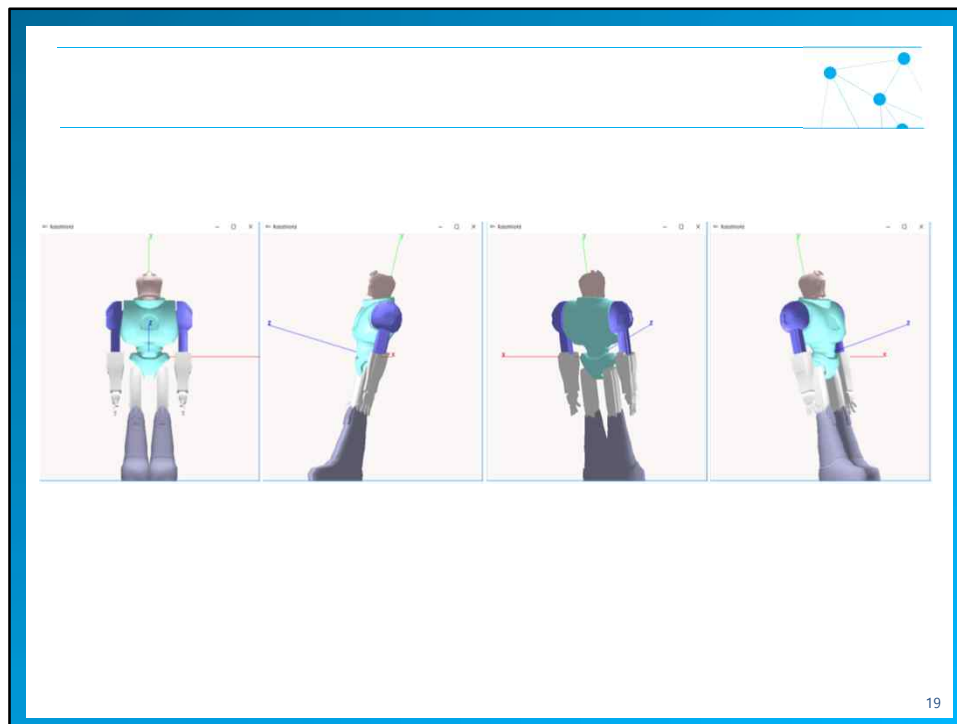


```

7. static bool bCamRot = false;
8-24. // 프로그램 6.1과 동일
25. void timerCallback(int tid) {
26.     if (bCamRot) {
27.         RotY += 5;
28.         if (RotY > 360) RotY = 0;
29.         glutTimerFunc(50, timerCallback, 1);
30.         glutPostRedisplay();
31.     }
32. }
33. void keyboard(unsigned char key, int x, int y) {
34.     // 프로그램 6.1의 함수와 동일. 31행 앞에 다음 코드 추가
35.     else if (key == 'v') {
36.         bCamRot = !bCamRot;
37.         glutTimerFunc(100, timerCallback, 1);
38.     }
39.     glutPostRedisplay();

```

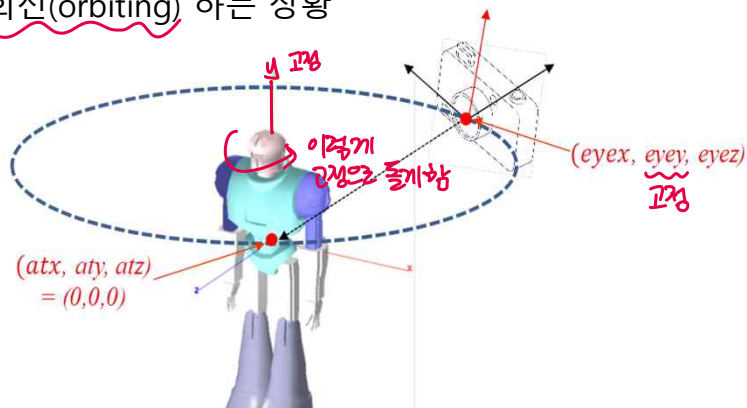
18



19

## gluLookAt() 사용한 시점변환 애니메이션

- 카메라가 일정한 높이에서 몸통 좌표계 원점을 보면서 회전(orbiting) 하는 상황



20

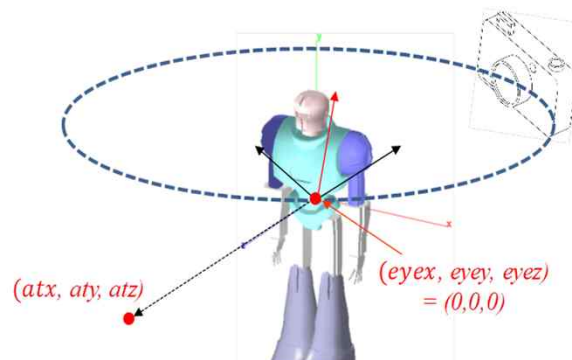
## gluLookAt() 파라미터 설정 1

- (eyex, eyey, eyez)
  - eyey는 고정
  - eyex와 eyez는 현재 각도 RotY의  $\sin()$  및  $\cos()$  값에 비례
- (atx, aty, atz)
  - 몸통 중심을 본다면 (0,0,0)으로 설정
- (upx, upy, upz)
  - Y 축 방향으로 설정하는 것이 편리  $\rightarrow (0,1,0)$
- 문제가 있음. Why?
  - 가시 부피(View Volume) 때문에 로봇이 화면에서 사라질 수 있음  $\rightarrow$  7장 내용 히든라이프가 없어졌다 생각다 하는 문제가 있음
  - 해결 방법  $\rightarrow$  (eyex, eyey, eyez)를 원점으로 고정


21

## gluLookAt() 파라미터 설정 2

- 해결  $\rightarrow$
- 시점 좌표계 원점을 전역좌표계와 일치시키고 at 좌표를 회전하는 상황의 애니메이션
    - (eyex, eyey, eyez) = (0,0,0)
    - (atx, aty, atz) = ( $\sin(\text{RotY})$ ,  $0.01 * \text{RotX}$ ,  $\cos(\text{RotY})$ )



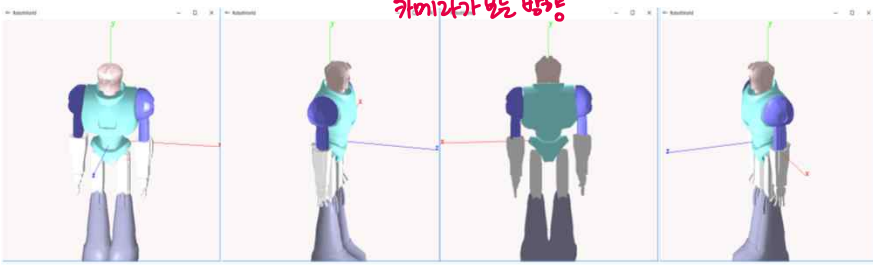
22



카메라 원점

`gluLookAt(0,0,0, SIN(RotY), 0.01*RotX, COS(RotY), 0, 1, 0);`

카메라가 보는 방향



23



## 6.4 로봇의 애니메이션: 달리는 로봇

- 계층적 모델과 애니메이션
- 동작의 설계와 구현

24

## 계층적 모델과 애니메이션



- 애니메이션을 위해서는 이들 관절의 각도나 위치를 시간에 따라 변경하면서 동작을 만들어야 함
- **운동학(Kinematics)** 분야에서는 모형의 각 부분에 대한 위치를 관절각으로만 표현하는 방법을 연구
  - 관절각의 변화를 지정 → 각 부품의 위치 계산 → 동작
  - 이때 보통 관성(inertia)이나 마찰(friction) 등의 효과는 무시
  - 순방향 / 역방향

25

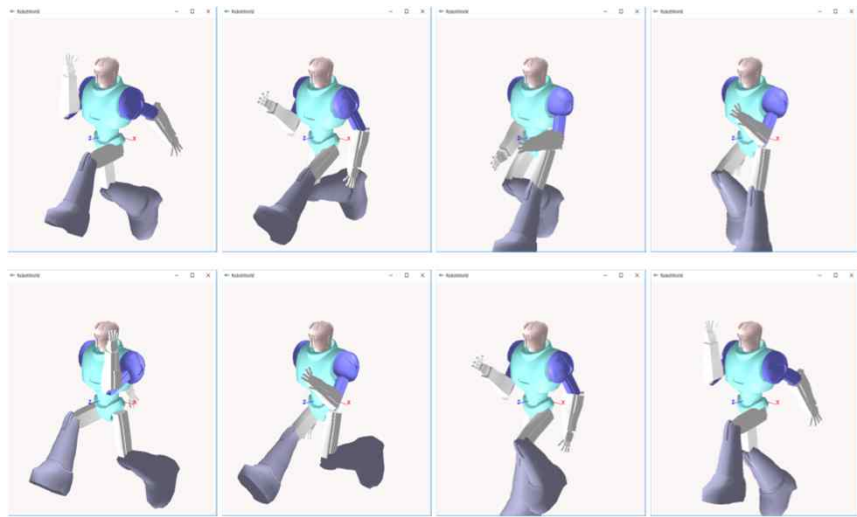


### 정방향 순방향 운동학(Forward Kinematics)

- 계층구조의 상위관절에서 하위관절로 내려오면서
- 각 관절에 사용자가 직접 필요한 만큼의 회전을 적용 → 동작
- 사용자가 원하는 대로 동작 구현 가능
- 매우 번거로운 작업
- **역방향 운동학(Inverse Kinematics)**
  - 계층구조의 맨 아래에 있는 부품의 위치만 사용자가 지정
  - 상위관절의 움직임을 컴퓨터가 계산하는 방법
  - 예) 팔 동작의 경우 손의 위치만 사용자가 지정하면 나머지 관절의 회전각 계산은 컴퓨터가 함 컴퓨터
  - 편리하기는 하지만 팔꿈치가 반대방향으로 꺾이는 등과 같은 부자연스러운 동작을 만들 수 있음.

26

## Lab: 달리는 로봇



27

## 애니메이션을 위한 코드 구조

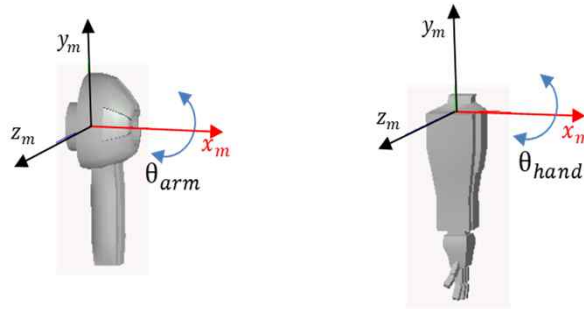
```

1. class Robot {
2.     ... // 프로그램 5.4의 로봇 클래스 모든 코드 추가
3.     void draw() {
4.         glPushMatrix();
5.         animateBody(); // 몸통 애니메이션: 모델변환
6.         Body.draw(0.5, 0.8, 0.8, scale, true);
7.         glPopMatrix();
8.         glTranslated(0.0, 0.45, -0.07);
9.         glScalef(1.1f, 1.1f, 1.1f);
10.        animateHead(); // 머리 애니메이션: 모델변환
11.        Head.draw(0.8, 0.7, 0.7, scale);
12.        glPopMatrix();
13.        ... // 다른 부품에도 동일한 형태의 코드 삽입
14.    }
15.    void animateBody() {...} // 몸통을 위한 애니메이션 코드
16.    ... // 모든 부품의 애니메이션 함수 추가
17. };
    
```

28

## 동작의 설계: 팔

- 팔 동작을 위한 피벗(pivot) 좌표 및 회전 축



29

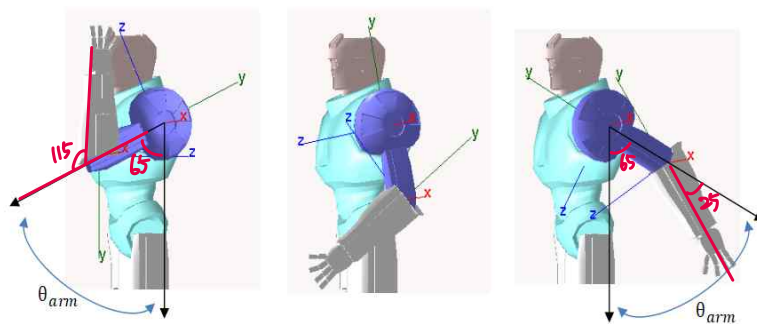
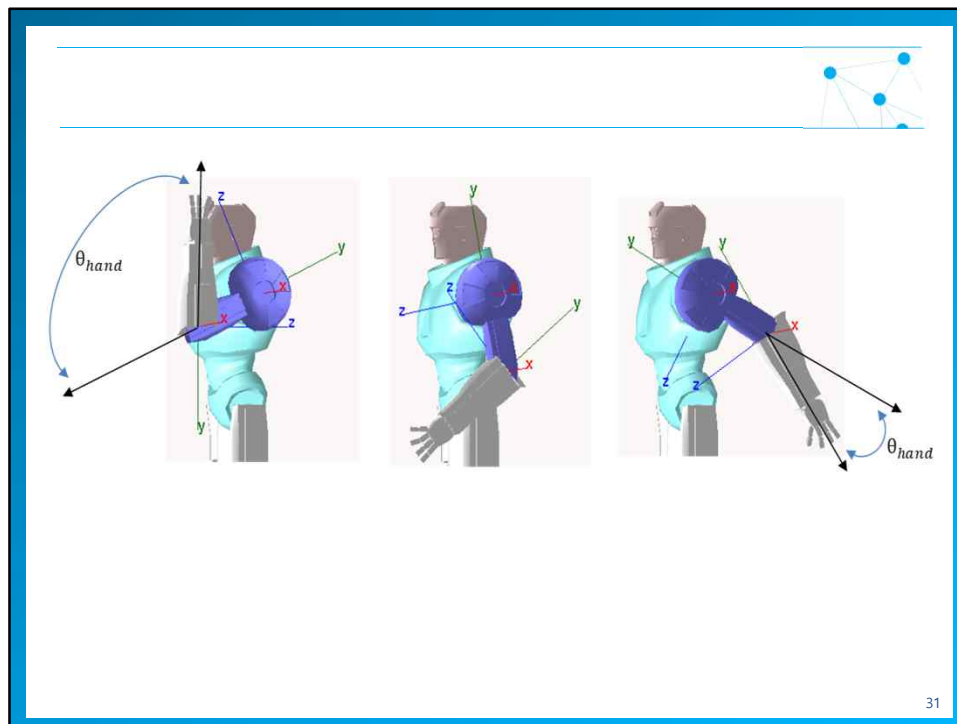


그림 6.9 팔 동작 모델링:  $\theta_{arm} = -65^\circ \sim +65^\circ$ ,  $\theta_{hand} = 25^\circ \sim 115^\circ$

30



31

- 동작(각도)의 변화량?
  - 팔의 각도가 일정한 크기로 증가하거나 감소되지 않음
  - 가운데에서 크고, 양쪽 끝에서 최소 → **삼각함수!**

$$\theta_{arm}(t) = \cos(t) \cdot 65^\circ$$

$$\theta_{hand}(t) = \cos(t) \cdot 45^\circ + 70^\circ$$

*Hand note: 70 ~ 115, 90-45=45*

*각각 2중성*

```
void animateLeftArm() { glRotatef(cos_t*65, 1, 0, 0); }
void animateLeftHand() { glRotatef(cos_t*45 + 70, 1, 0, 0); }
```

32



## 동작의 설계: 다리

- 다리도 같은 방법으로 처리

$$\theta_{leg}(t) = -\cos(t) \cdot 45^\circ + 35^\circ \quad (-10^\circ \sim 80^\circ)$$

$$\theta_{foot}(t) = -\cos(t-90) \cdot 50^\circ - 55^\circ = -\sin(t) \cdot 50^\circ - 55^\circ$$

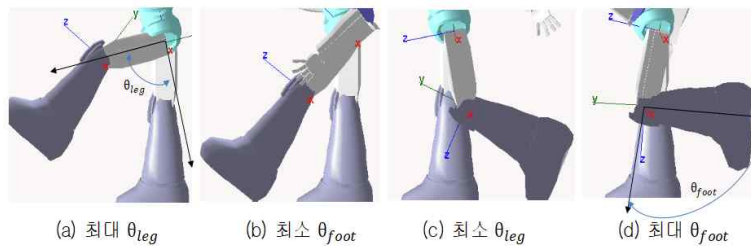


그림 6.10 다리 동작 모델링( $\theta_{leg} = -10^\circ \sim +80^\circ$ ,  $\theta_{foot} = -5^\circ \sim 105^\circ$ )  $\theta_{leg}$  과  $\theta_{foot}$ 의 사이클에 90도 위상차가 있음.

33

## 보다 실감나는 동작

- 몸통
  - 달리는 동작에는 보통 몸통 전체가 좌우로도 움직임.
  - 이를 위해 이동과 회전 변환 필요.
  - 몸통의 상하 움직임
    - y방향의 이동 변환
    - 한 사이클에서 상하 이동은 두 번
  - 몸통의 상하 회전
    - 달릴 때 몸이 앞으로 살짝 굽었다 펴지는 과정을 반복
    - 회전축은 x축, 한 사이클 동안에 두 번의 회전 동작
  - 몸통의 좌우 회전은 y축이 중심
    - 왼발을 디딜 때에는 몸통의 왼쪽이 앞으로 가야 하고 오른발은 그 반대이다.

34



- 머리
  - 달릴 때 머리로 보통 좌우로 약간씩 회전
  - y축이 중심
- 팔과 다리
  - 앞에서 설계한 팔과 다리의 동작에 약간의 비틀림을 추가
  - 이러한 비틀림은 y축을 중심으로 한 회전으로 구현

35

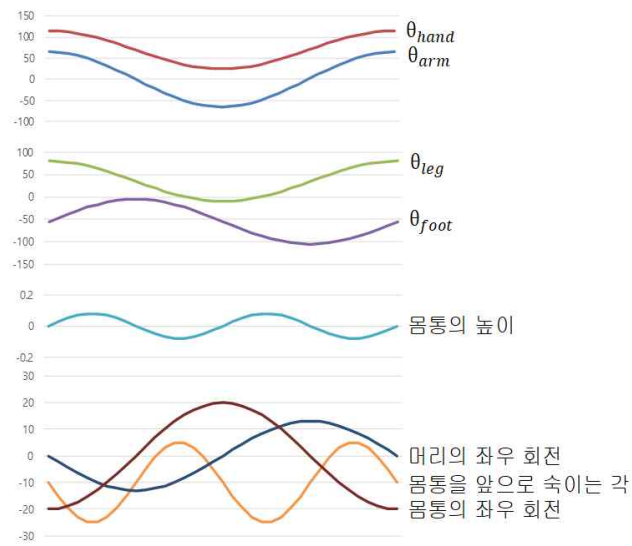
## Robot::run() 함수 구조



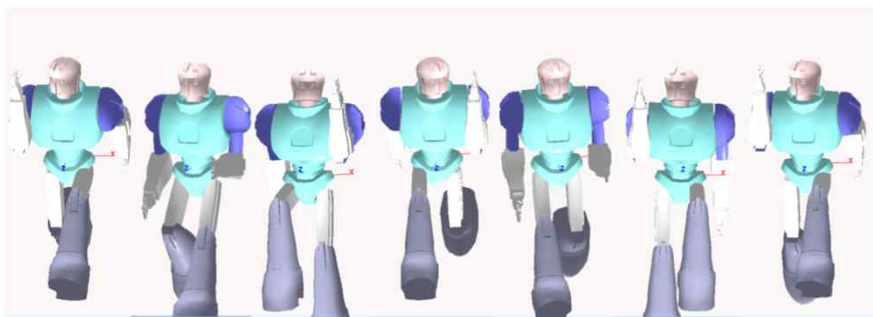
```
11. void stop() { bRun = false; }
12. void run() {
13.     bRun = true;
14.     tAngle += 20;
15.     if (tAngle >= 360)           // tAngle: 0~360
16.         tAngle -= 360;
17.     sin_t = SIN(tAngle);         // degree 각에 대한 sin
18.     cos_t = COS(tAngle);         // degree 각에 대한 cos
19.     sin_t2 = SIN(2 * tAngle);    // 2배 빨리 움직임
20. }
```

36

## 달리는 로봇: 한 주기 동안의 각도 및 이동 곡선



37



38

## 더 자연스러운 애니메이션?



- 모션캡처 데이터를 이용한 애니메이션
- BVH파일 적용

39

## 6장 연습문제



- [Lab 6-1] Lab 5-3에서 구현한 자신만의 로봇에 앞에서 구현한 달리는 동작을 적용해 보라. 로봇의 각 관절의 길이가 다르고 부품의 개수도 다를 것이므로 동작을 위한 파라미터들을 적절히 조절해야 할 것이다. 최대한 자연스러운 동작이 되도록 구현해 보라.
- [Lab 6-2] 뛰는 동작이 아니라 걷는 동작이나 팔을 들어 올리는 동작 등 자신만의 동작을 순방향 운동학을 이용해 만들어 보라.

40



- [Lab 6-3] 로봇 부품들이 하나씩 합체되는 과정을 애니메이션으로 구현해 보라.
- [Lab 6-4] 만들어진 동작에 카메라 움직임을 더해 애니메이션을 만들어 보라. 카메라는 6.3절과 같이 원형으로 움직일 수도 있지만 임의의 궤적을 지정해 움직이도록 할 수도 있다. 이 경우 이러한 카메라 궤적은 매우 연속적인 곡선을 이루도록 해야 장면이 흔들리거나 부자연스럽지 않다. 보통 이를 위해 스플라인 곡선을 이용한다.

41



**감사합니다!**

42