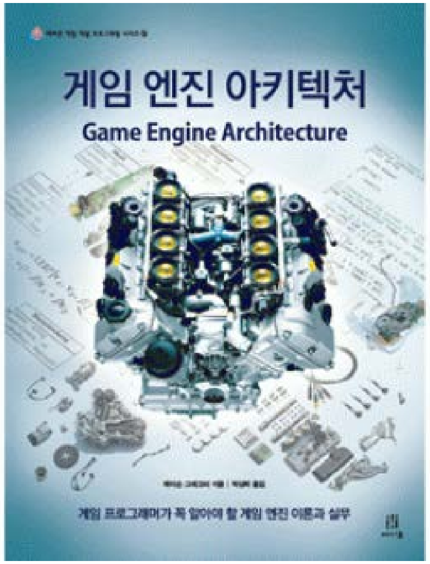
Computer Graphics: Timing

Dept. of Game Software
Yejin Kim

Overview

- Game Loop
- Timing
- Tutorials



*Game Engine Architecture (Ch. 7)

Game Loop

- Rendering loop in games and OS GUI
  - Redraw in Windows apps
    - Mostly static screen
    - Small changes in parts of screen
    - Rectangle invalidation method
      - Only redraw the modified parts
  - Redraw in games
    - Mostly dynamic screen
    - Big changes from camera movement
    - Rendering loop required due to the real-time sequence of static images

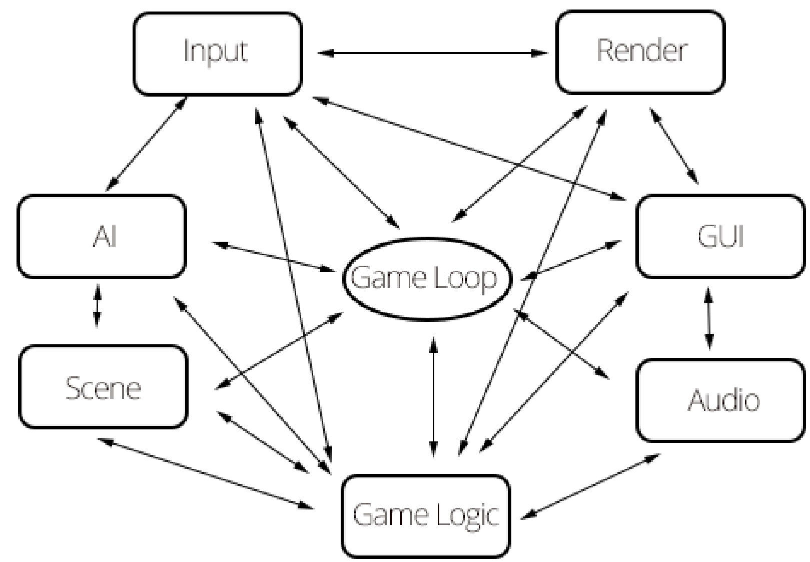
Game Loop

- General architecture of rendering loop

```
while(!quit)
{
            // Update camera transform based on user input, or predefined scenario
            updateCamera();

            // Update positions, orientations and other visual states of the elements
            updateSceneElements();

            // Render a still frame into the "back buffer"
            renderScene();

            // Swap the back buffer with the front buffer
            swapBuffers();
}
```

Game Loop

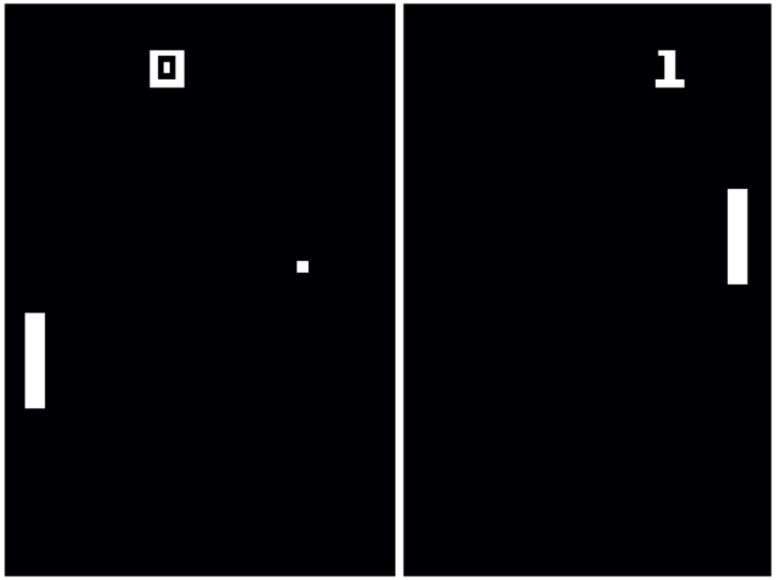
- Subsystems in games
  - Device IO
  - Rendering
  - Animation
  - Physical simulation
  - Artificial intelligence
  - Multiplayer networking
  - Audio



- Periodic update → Game loop
  - Different update period for each subsystem
    - Rendering and animation: 30~60Hz
    - Physical simulation: >120Hz
    - Behaviors in AI: 1~2Hz

Game Loop

- Pong game
  - Atari arcade game: http://www.pong-story.com/arcade.htm
  - Simple, but exciting..!



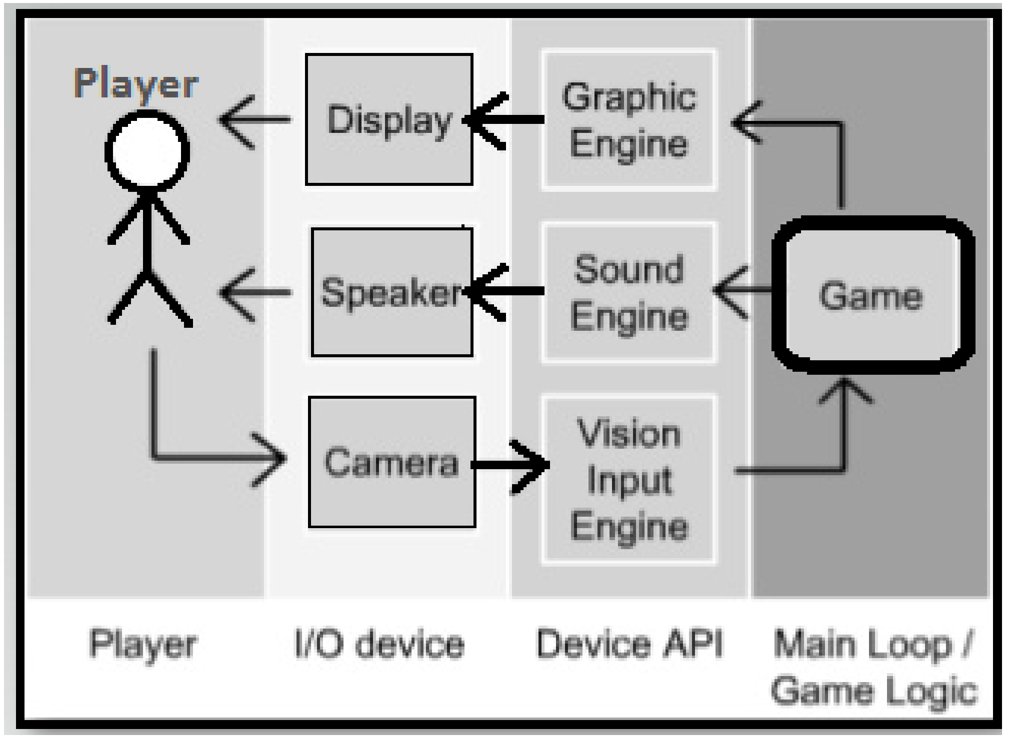
Game Loop

- Pong pseudocode(의사코드)

```
void main()                         // pong
{
    initGame();                     // Init HWs

    while(true) {                   // Game Loop
        readInterfaceDevices();

        if (quitButtonPressed())
            break;

        movePaddles();
        moveBall();
        collideAndBounceBall();

        if (ballImpactedSide(LEFT_PLAYER) {
            incrementScore(RIGHT_PLAYER);
            resetBall();
        }

        if (ballImpactedSide(RIGHT_PLAYER) {
            incrementScore(LEFT_PLAYER);
            resetBall();
        }
        renderPlayfield();          // Render all contents
    }
}
```

Game Loop

- Game loop architecture
  1. Windows message pump(윈도우 메시지 펌프)
  2. Callback-driven framework(콜백 주도 프레임워크)
  3. Event-based update(이벤트 기반 업데이트)



Game Loop

- Windows message pump
  - Routine for Windows-based apps handling OS messages
    - Follow Windows rules
    - Execute game loop after handle Windows message

```
// Message Pump for a game
while (true)
{
	// Service all pending windows messages
	while (PeekMessage(&msg, NULL, 0, 0)>0)
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}

	// If no more windows messages,  run one iteration of the game loop
	RunOneIterationOfGameLoop();
}
```

Game Loop

- Framework
  - Template for implementing application
    - Can NOT modify the overall program flow
    - Mostly, empty functions (e.g. virtual functions)
- Callback-driven framework
  - Completing the framework by overriding callback functions
    - Callback function: a function call is decided by the system, not a programmer
  - 비동기처리가 많은 경우 그 처리시간이 오래 걸림
  - System이 결과 값을 callback 함수를 호출하여 넘겨줌

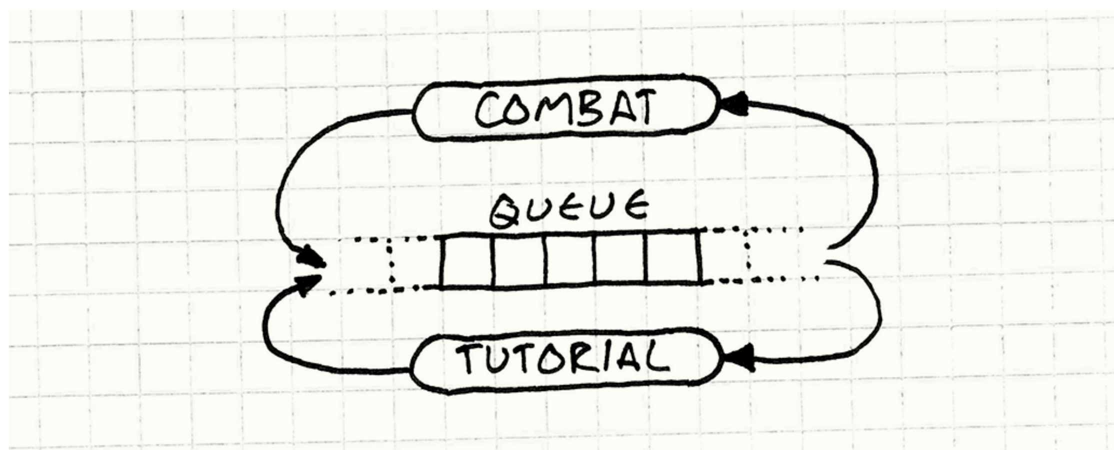
Game Loop

- Callback-driven framework

```
#include <windows.h>

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow)
{
	…

	while(GetMessage(&Message, 0, 0, 0)) {
		TranslateMessage(&Message);
		DispatchMessage(&Message);
	}

	return Message.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT iMessage, WPARAM wParam, LPARAM lParam) {
	switch(iMessage) {
		case WM_DESTROY:
		PostQuitMessage(0);
		return 0;
	}

	return(DefWindowProc(hWnd, iMessage, wParam, lParam));
}
```

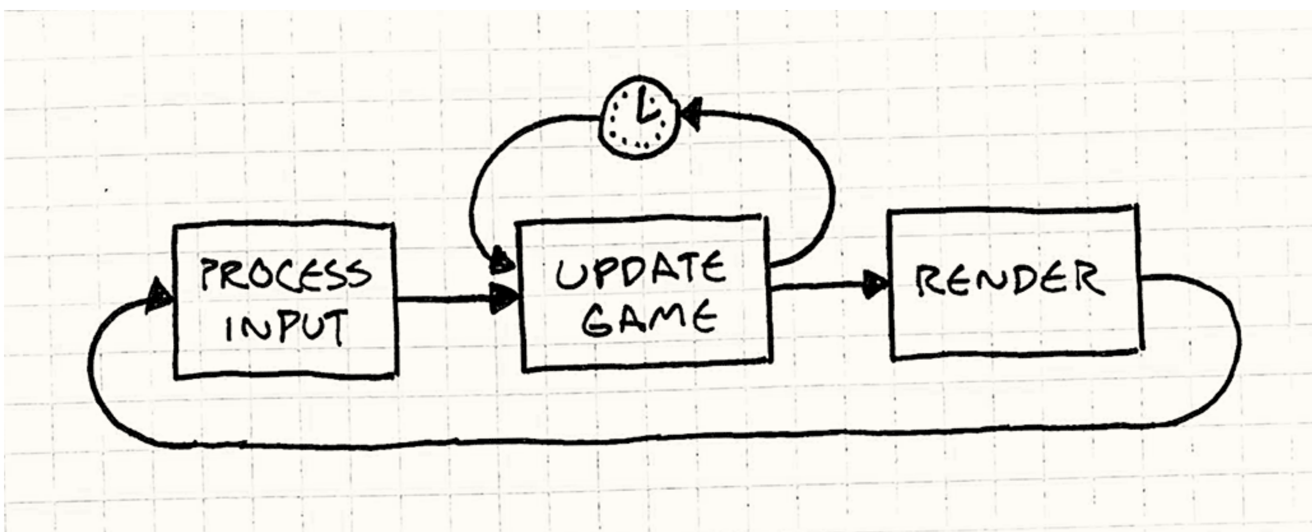
Game Loop

- Event
  - Certain changes in game states
  - e.g. key input, explosion start, alarmed by enemy, etc.
- Event-based update
  - Update subsystem periodically using event system
    - e.g. Dispatch an event every 1/30s
    - Event queue for future events



Timing

- Time in games
  - Real-time, dynamic, and interactive computer simulation
  - Game에서 시간은 매우 중요한 역할
  - Times in game engine
    - Game time
    - Animation time
    - Update time
    - Real (CPU) time



Timing

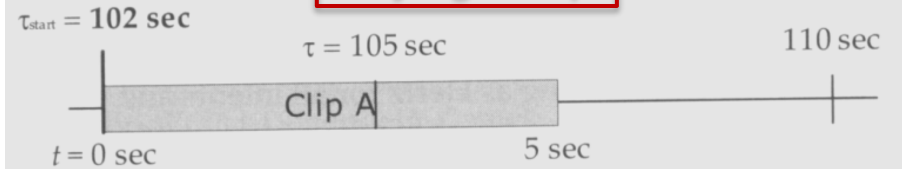
- Game time
  - Game에서 측정하거나 필요한 시간
  - 일반적으로 real time과 일치
  - 실시간과 같을 필요 없음
    - Slow 또는 fast 동작 재생
    - 일시적으로 game 정지
    - Debugging시 도움: game loop는 계속 실행

- Real time
  - CPU의 정밀 time register 값으로 측정한 시간
  - 원점: CPU의 power가 처음 들어온 순간 혹은 reset된 순간
  - 단위: CPU cycle → 초단위로 변환하여 사용

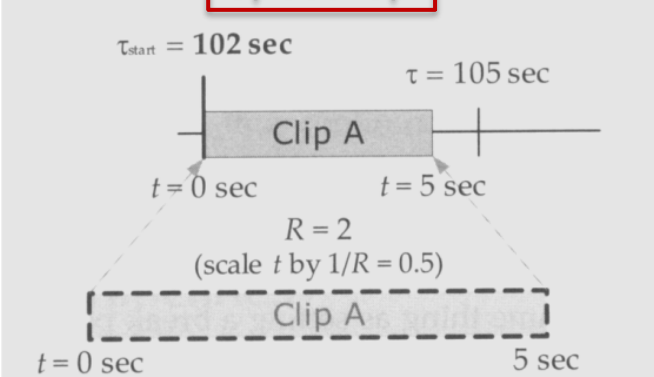
Timing

- Virtual timeline: local 및 global timeline
  - 각 animation이나 audio clip은 다른 timeline을 가질 수 있음
    - 각 클립은 t = 0로 정의되는 원점에서 시작
    - Playback(재생)시 속도 조절이 용이함: speed up 및 down
    - Play backward(역 재생)이 용이함

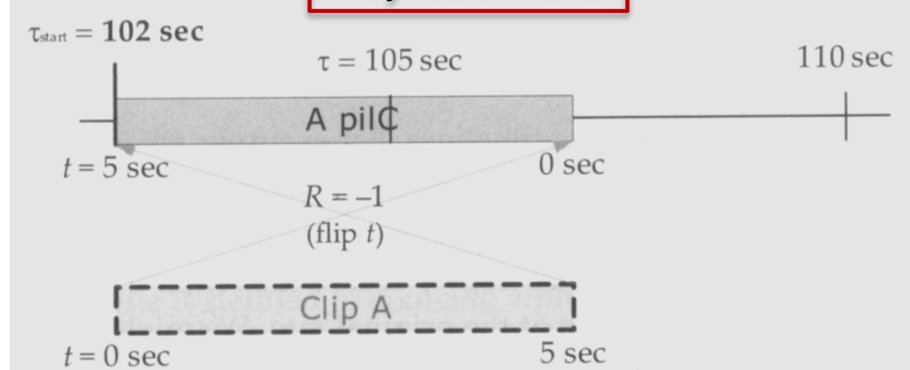
Playing a clip



Speed up



Play backward



Timing

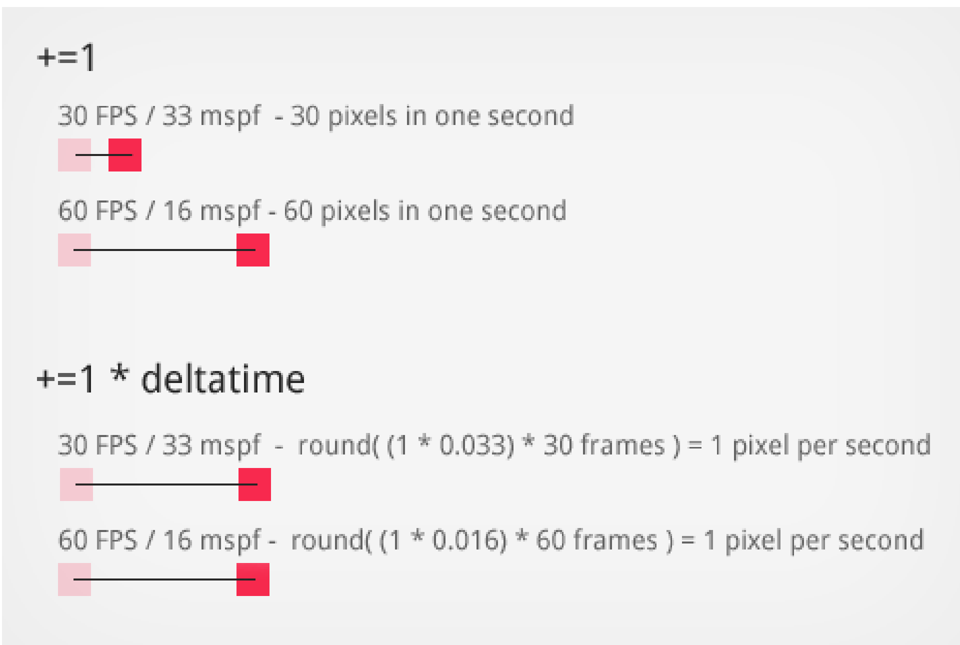
- Frame rate: f
  - Static frame을 연속적으로 얼마나 빠르게 보여주는 단위
  - Hertz (Hz): 초당 cycle(주기)의 수
  - Frames per second (FPS)
    - Game과 영상 산업에 주로 쓰이고, Hz와 같은 개념
    - NTSC 방식 refresh rate(재생 빈도): 30 or 60 FPS (북미, 일본)
    - PAL or SECAM 방식: 50 FPS (유럽)
- Delta time: Δt
  - Frame간 시간의 차이
    - Also called frame time or time delta
    - Frame rate의 역수: $1/f$
      - E.g. 30 FPS = 1/30 sec = 0.00333s = 33.3 ms

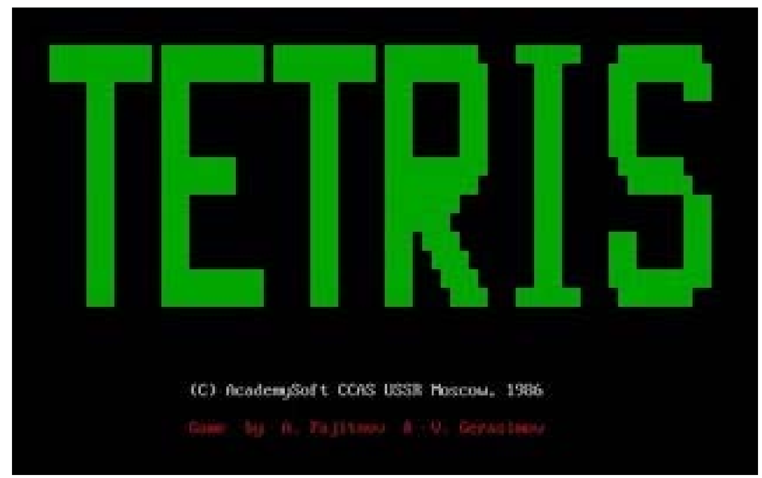
Timing

- Delta time: frame rate와 speed의 관계
  - Speed: v
    - Meters per second (or pixels per second for 2D games)
  - e.g. 우주선을 움직인다면,
    위치의 변화: $\Delta x = v \Delta t$
    주어진 현재 위치가 x_1일 때, 다음 프레임에서의 위치:
    $x_2 = x_1 + v \Delta t$
  - Game에서 모든 object의 움직임이 Δt 에 영향을 받음
  - 적합한 Δt 는 어떻게 구할 수 있을까?

Timing

- Delta time: Δt 대신 일정한 fixed time(고정된 시간) 사용시
  - Frame당 meter(혹은 pixel 등)로 직접 지정
  - CPU의 성능에 따라 전체 game speed가 틀려짐
    - e.g. 빠른 CPU → 게임이 빨라짐
    - CPU의 speed에 dependent(종속적) 게임





Timing

- Delta time: elapsed time(경과 시간)에 따른 update
  - CPU 성능에 independent(독립적인)한 *Δt* 측정
  - CPU 시간을 정밀 타이머로 2번 측정
    - Frame 시작에 한번, 끝에 한번
    - 측정한 delta 값은 필요한 모든 subsystem에서 사용
  - 현재 대부분의 game engine에서 사용
    - 현재 frame에서 측정한 *Δt* 을 다음 프레임의 *Δt 로* 사용
      - *"Past performance is not a guarantee of future results"*
    - Frame-rate spike: 갑작스런 frame rate의 감소 증상
      - e.g. Physics system의 안정적 update: 33 ms
        만약, frame이 떨어져 57 ms가 걸린 경우, physics system이 다음 frame때 2번 simulation 됨 → 상황 악화!

Timing

- Delta time: elapsed time(경과 시간)에 따른 update
  - 고정 frame rate 사용
    - 모든 frame의 시간을 33.3 ms (30 FPS) 또는 16.6 ms (60 FPS)로 고정
    - 현재 frame의 시간을 측정 후, 목표 시간보다
      짧으면, 기다림
      길면, 그냥 감수하고 한 frame 더 기다림
    - 전체적으로 *평균 FPS ≈ 목표 FPS* 에만 동작
  - 실행한 average time(평균 시간) 사용
    - 적은 수의 frame 시간을 평균 내서 다음 frame의 예측 Δt 로 사용
    - 변화하는 frame rate에 대응할 수 있음
    - 갑작스런 frame rate의 감소 현상(frame-rate spike) 완화

Tutorials

- Rendering Information
- First Person Camera
- Free Look Camera
- DirectX Input and Sound

5-1 Rendering Information

- Timer: Real-time estimation
  - System time을 알려주는 함수 사용 (C언어)

    time();

    - 현대 game에선 정확성이 부족한 경우가 많음
    - e.g. The number of *seconds* since midnight, Jan 1, 1970
  - High-resolution timer (정밀 타이머) 사용 (Win32 APIs)

    QueryPerformanceCounter(); // read the counter (64bit integer)

    QueryPerformanceFrequency(); // number of cycles per second

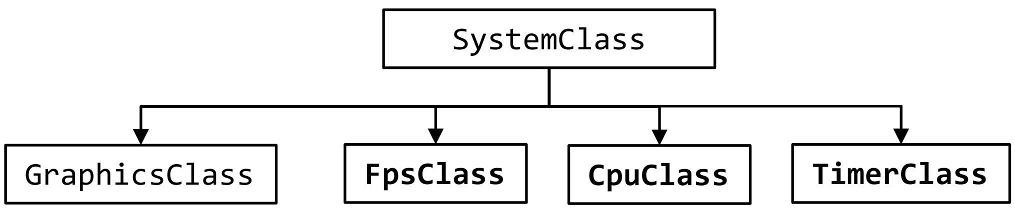
    - CPU에 power가 reset된 시점부터 초당 cycle의 수를 측정
    - e.g. 3GHz CPU의 경우 초당 30억분의 1로 시간을 나눌 수 있음

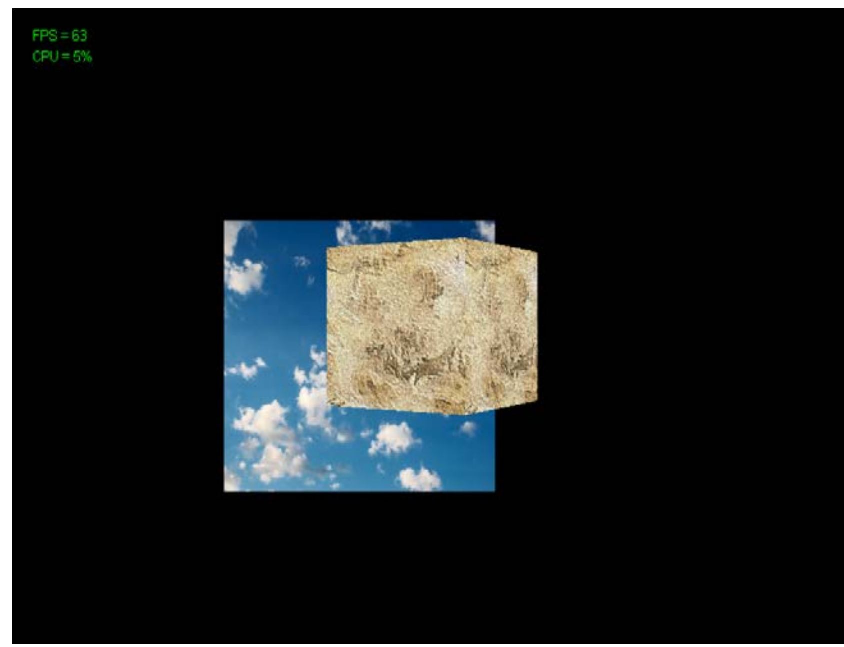
      정밀도: 1/(30억) = 3.33×10^{-10} sec = 0.333 ns

      64-bit int register를 timer로 사용시 reset되는데 195년 걸림

5-1 Rendering Information

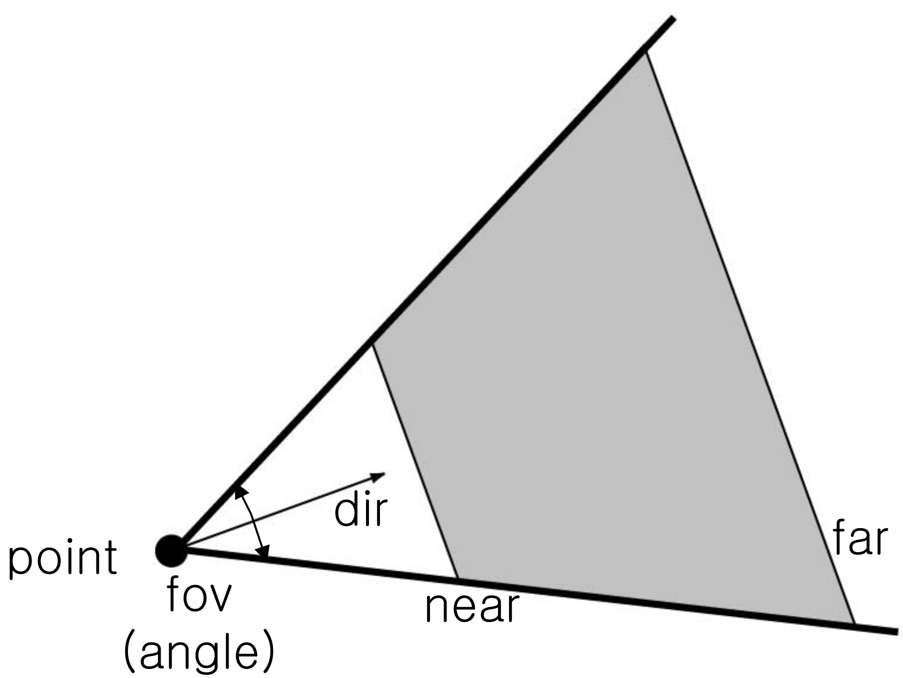
- Draw FPS and CPU usage on screen
  - TimerClass: a high precision timer that measures the exact time between frames of execution
  - FpsClass: counts and updates frame numbers
  - CpuClass: estimates the total CPU usage

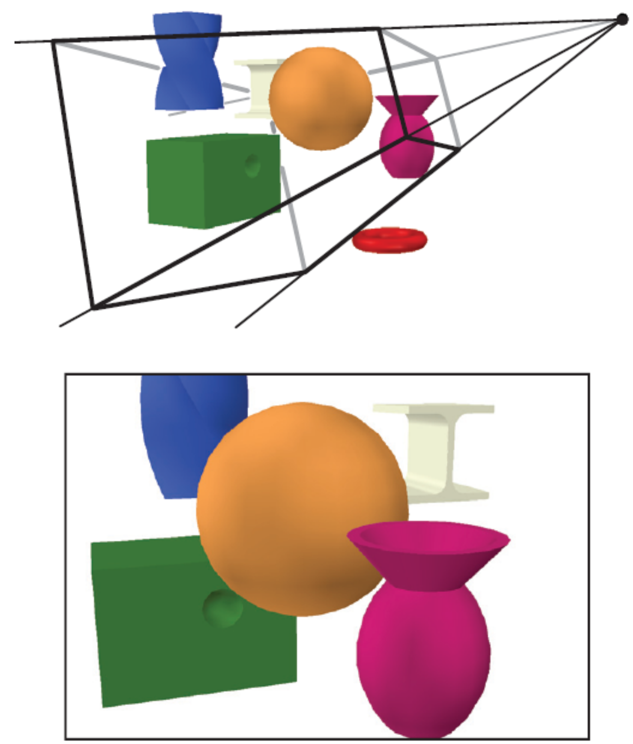




5-2 First Person Camera

- Creating the first person camera (*BraynzarSoft)
  - Defined by position, direction vector, up vector, field of view, near and far plane
  - Create image of geometry inside gray region
  - Used by OpenGL, DirectX, ray tracing, etc

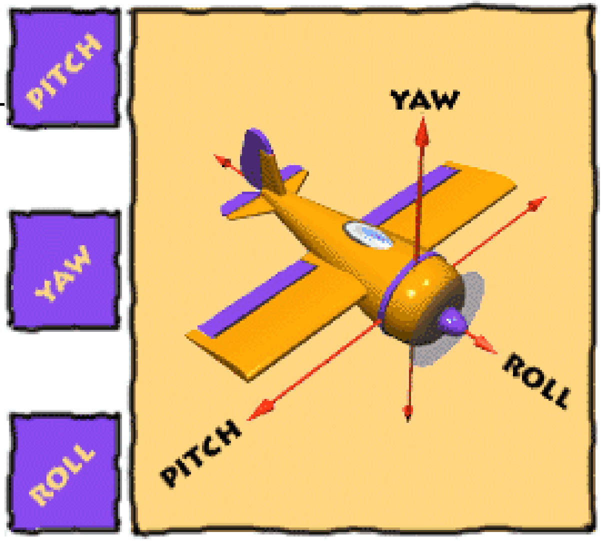




5-2 First Person Camera

- Creating the first person camera (*BraynzarSoft)

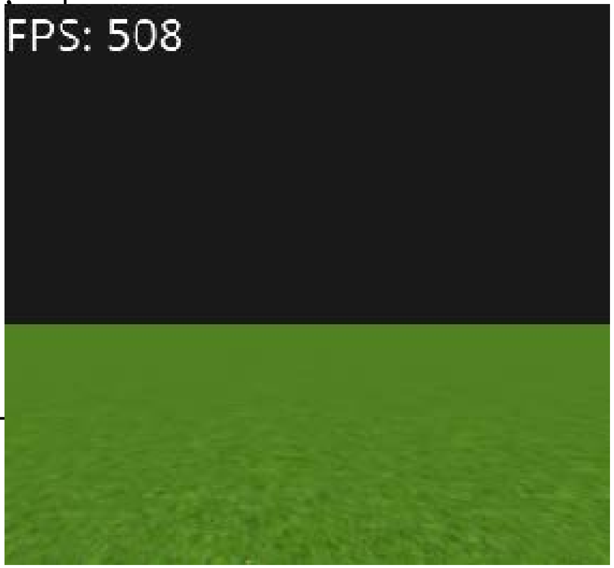
```
XMVECTOR DefaultForward = XMVectorSet(0.0f,0.0f,1.0f, 0.0f);// the forward direction in the world
XMVECTOR DefaultRight = XMVectorSet(1.0f,0.0f,0.0f, 0.0f);  // the right direction in the world
XMVECTOR camForward = XMVectorSet(0.0f,0.0f,1.0f, 0.0f);    // the forward direction of the camera
XMVECTOR camRight = XMVectorSet(1.0f,0.0f,0.0f, 0.0f);      // the right direction of the camera

XMMATRIX camRotationMatrix;         // the rotation matrix of the camera
XMMATRIX groundWorld;               // the world matrix of the ground plane

float moveLeftRight = 0.0f;         // to move the camera strafe right/left
float moveBackForward = 0.0f;       // to move the camera forward/backward

float camYaw = 0.0f;                // rotation around the y-axis
float camPitch = 0.0f;              // rotation around the x-axis
```



5-2 First Person Camera

- Creating the first person camera (*BraynzarSoft)

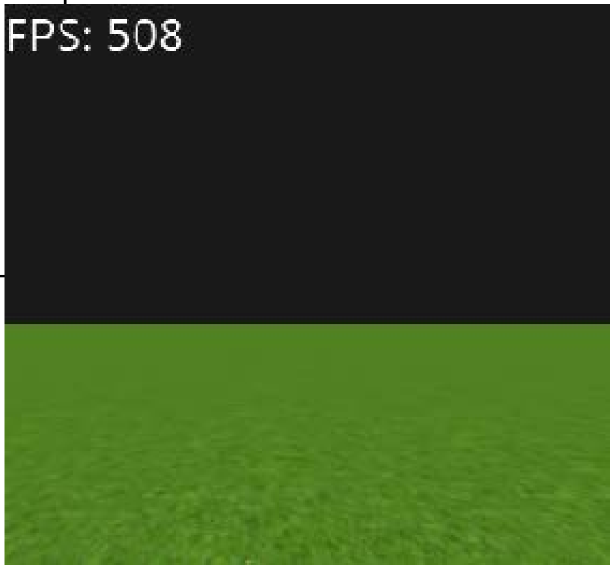
```
void UpdateCamera() {
	// Rotating the camera
	camRotationMatrix = XMMatrixRotationRollPitchYaw(camPitch, camYaw, 0);
	camTarget = XMVector3TransformCoord(DefaultForward, camRotationMatrix );
	camTarget = XMVector3Normalize(camTarget);

	// Restricting the camera rotation around the y-axis
	XMMATRIX RotateYTempMatrix;
	RotateYTempMatrix = XMMatrixRotationY(camYaw);

	// Updating the camera's right, up, and forward vectors
	camRight = XMVector3TransformCoord(DefaultRight, RotateYTempMatrix);
	camUp = XMVector3TransformCoord(camUp, RotateYTempMatrix);
	camForward = XMVector3TransformCoord(DefaultForward, RotateYTempMatrix);

	// Moving the camera
	camPosition += moveLeftRight*camRight;
	camPosition += moveBackForward*camForward;
	moveLeftRight = 0.0f;
	moveBackForward = 0.0f;

	// Updating the camera matrix
	camTarget = camPosition + camTarget;
	camView = XMMatrixLookAtLH( camPosition, camTarget, camUp );
}
```



5-3 Free Look Camera

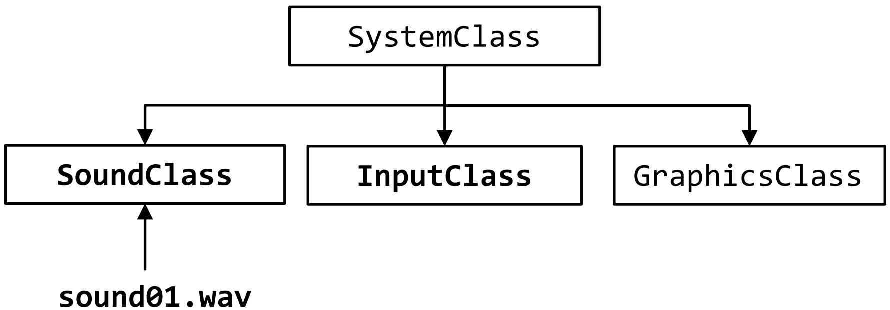
- Creating the free look camera (*BraynzarSoft)

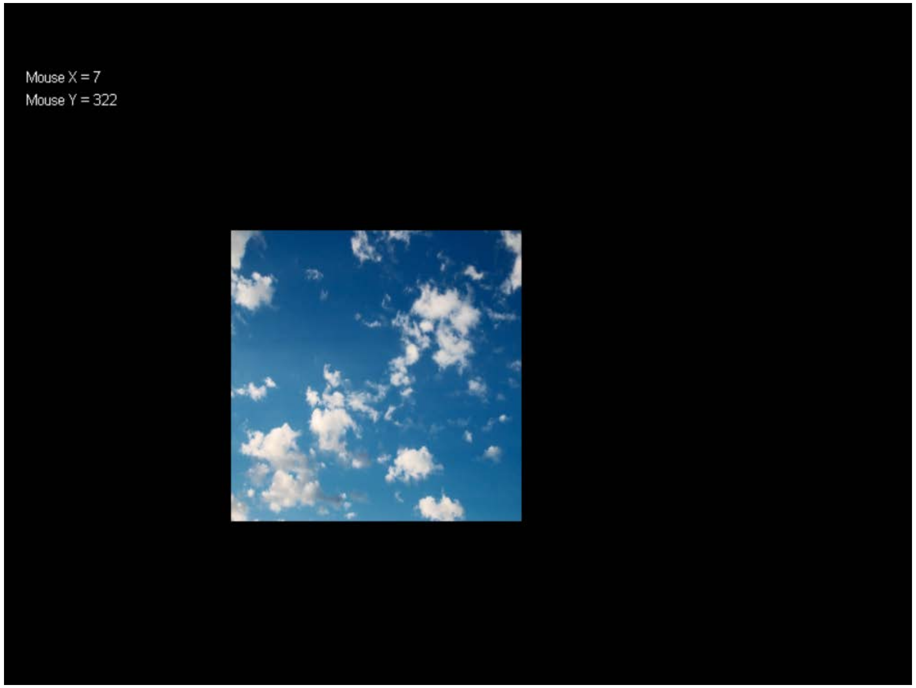
```
void UpdateCamera() {
	// Rotating the camera
	camRotationMatrix = XMMatrixRotationRollPitchYaw(camPitch, camYaw, 0);
	camTarget = XMVector3TransformCoord(DefaultForward, camRotationMatrix );
	camTarget = XMVector3Normalize(camTarget);

	// Updating the camera's right, up, and forward vectors
	camRight = XMVector3TransformCoord(DefaultRight, camRotationMatrix);
	camForward = XMVector3TransformCoord(DefaultForward, camRotationMatrix);
	camUp = XMVector3Cross(camForward, camRight);

	// Moving the camera
	camPosition += moveLeftRight*camRight;
	camPosition += moveBackForward*camForward;
	moveLeftRight = 0.0f;
	moveBackForward = 0.0f;

	// Updating the camera matrix
	camTarget = camPosition + camTarget;
	camView = XMMatrixLookAtLH( camPosition, camTarget, camUp );
}
```



5-4 Direct Input and Sound

- Adding Direct input and sound to the framework
  - InputClass: accepts input devices
  - SoundClass: loads and plays WAV audio files





5-4 Direct Input and Sound

- Initializing Direct input devices: keyboard and mouse
  - DirectInput8Create()
  - IDirectInput8::CreateDevice()
  - IDrectInputDevice8::SetDataFormat()
  - IDrectInputDevice8::SetCooerativeLevel()
  - IDrectInputDevice8::Acquire()
  - IDrectInputDevice8::Unacquire()
  - IDrectInputDevice8::Release()

- Use WAV audio files
  - Should use WAV format: 44.1KHz, 16bit, 2 channels
  - Do **not** use a web-based converter for sound files: MP3 → WAV

References

- Wikipedia
  - www.wikipedia.org
- Introduction to DirectX 11
  - www.3dgep.com/introduction-to-directx-11
- Raster Tek
  - www.ratertek.com
- Braynzar Soft
  - www.braynzarsoft.net)
- CS 445: Introduction to Computer Graphics *[Aaron Bloomield]*
  - www.cs.virginia.edu/~asb/teaching/cs445-fall06

Q & A