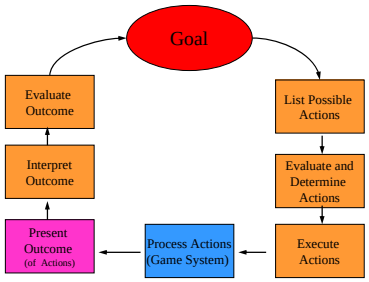
Game Rendering

Doron Nussbaum

Agenda

- Game Engine Review
- Direct X Review
- Graphics Engine

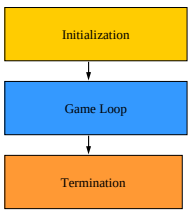
Stages of Game Interaction



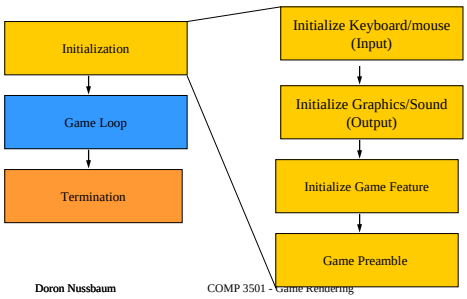
Game Engine

- Purpose:
  - Master - manages the game at all phases
- Is it in full control?
  - No - with OS being windows based
    - X, WidowsXX, Linux

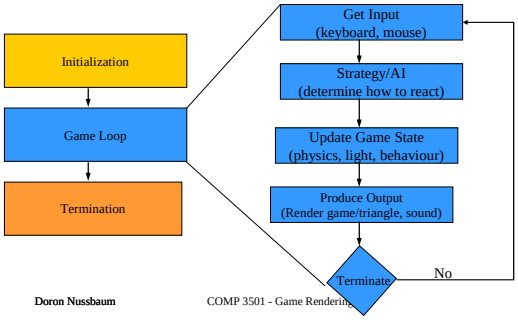
Game Engine



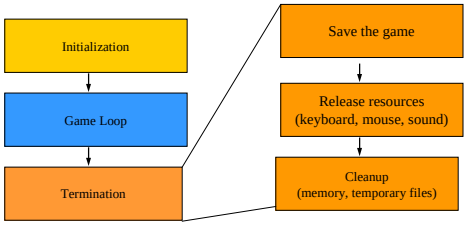
Game Engine



Game Engine



Game Engine



Engines

- What are engines?
  - Framework/components/subsystems that can be used to develop a game (software?)
  - Libraries of APIs
  - Middleware
- Purpose:
  - Saving development time
  - Non need to repeat work
  - Reusable components
  - Higher quality results (performance, accuracy…)
  - Reducing need for a large team of expert – can focus on we are good at

Game Engines and Senses

- Games (video games) can benefit from incorporating our senses.
- Movies – vision, hearing
- Games
  - Vision
  - Hearing
  - Feeling?
  - Smell?
  - Taste?
- Use devices and engines to enhance the experience

Game Engines and Senses

- Games
  - Sight/Vision –
    - 3D,
    - colours,
    - animation,
  - Hearing –
    - music
    - voice,
    - sounds (thunder, car motor)
  - Touch/Feeling –
    - haptic devices
  - Smell
    - devices that release scent
  - Taste
    - devices that release scent mimicking taste
    - devices that release some taste

Games vs. Movies

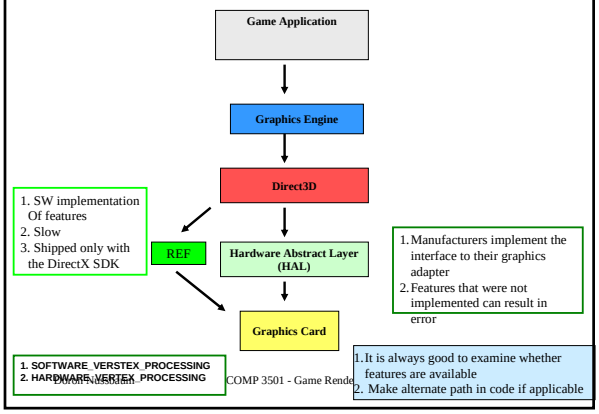
- Games differ from movies in several ways
  - Interaction – active role (non passive)
  - "Unpredicted" outcome
  - "Variation" of script
- Incorporating new devices and options
  - Movies mainly affect our vision, and hearing
  - New devices can enhance the experience
    - Physical activity (Nintendo, Microsoft)
- Where and how can other devices be used?
- What games or activities can be enhanced from using devices?

Engines

- In many cases engines enhance our experience
  - Operate on our senses
  - Logic
  - Expectations
- Engines
  - Physics – realistic results (e.g., car does not stop, falling off a cliff)
  - Sound – affects our hearing
  - Graphics – affects our vision
  - Haptic engine
  - AI – meets expectation of behaviour
  - Networking ?
  - …

Graphics Engine (last)

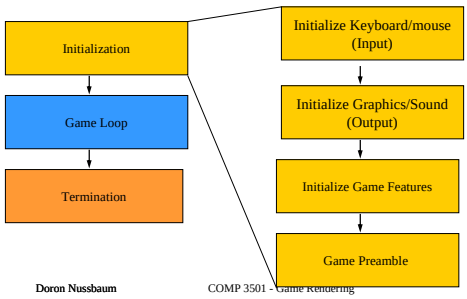
- Purpose
  - Designed to assist in rendering images and presenting them to the user
- Prime mechanism for delivering information to the player
  - Probably delivers 70-80 of information (e.g. reading)
  - Other than socializing (but may not be as true with SMS)
- Simple
  - Text
  - Images
    - Text based images
    - 2D images
    - 3D images
- Complex
  - Virtual reality – caves
  - Special effects



DirectX

- A collection of libraries that can be used to develop games.
  - Graphics – Direct3D
  - Input – DirectInput Xinput
  - Sound - DirectSound

Game Engine



Input

Mouse

- 2D coordinates
  - x, y coordinates of the screen
- Mouse buttons
  - Left
  - Right
  - Middle (scroll)
- What about 3D?

Keyboard

- Player's keyboard usage
  - Keys that were pressed
  - Implicitly – keys that were released (requires history)
  - Can use a combination of keys

Input Process

- Steps for obtaining input
  - Create an interface for the device
    - Tell windows which device to use
  - Setup access privileges
    - Telling windows who can access the device
    - What control does the program have
  - Setup the input data format
    - Telling DirectX how the communication will be done
  - Acquire the device
    - Obtaining access to the device
  - Read the data
    - Getting the data and responding
  - Unacquire the device
    - Giving up the device when not in need
  - Release the interface

Step 1 - Create an interface to the device

```
#include "dinput.h"

LPDIRECTINPUT8 din;  // A COM input interface (Direct8 object!!!)

// create the DirectX input interface
DirectInput8Create(hInstance, DRECTINPUT_VERSION, IID_IDirectInput8, (void **) &din, NULL);
```

```
// the com interface for the keyboard
LPDIRECTINPUTDEVICE8 dinKeyboard;
rc = din->CreateDevice(GUID_SysKeyBoard, &dinkeyboard, NULL);
```

```
// get the mouse device
LPDIRECTINPUTDEVICE8 dinMouse;
din->CreateDevice(GUID_SysMouse, &dinMouse, NULL);
```


Step 2 - Initialize the device

DISCL_BACKGROUND – Background access. The device can be acquired at any time.

DISCL_EXCLUSIVE – Exclusive access. Only the Application has access when device is acquired.

DISCL_FOREGROUND – Foreground access. Device is unacquired when window is in bg

DISCL_NONEXCLUSIVE - Nonexclusive access.

```
dinMouse->SetDataFormat(&c_dfDIMouse); // set the data format

dinMouse->SetCoopertiveLevel(hWnd, DISCL_NONEXLUSIVE, DISCL_BACKGROUND);

dinMouse->Acquire(); // acquire the mouse
```


Step 2 – Reading the Data

Set up the structures

Read the data

Use it

```
dinMouse->SetDataFormat(&c_dfDIMouse);
```

```
DIMOUSESTATE mouseState;

dinMouse->Acquire();  // acquire the mouse

dinMouse->GetDeviceState(sizeof(DIMOUSESTATE),  (void *) mouseState);
```

```
if (mouseState.rgbButtons[0] & 0x80)  rocket->fire();
```


Step 2 – Reading the Data

Set up the structures

Read the data

Use it

```
dinKeyboard->SetDataFormat(&c_dfKeyboard);
```

```
rc = dinKeyboard->GetDeviceState(256,keys);
while (rc != DI_OK) {
   switch (rc) {
      case DI_OK:
            break;
      case DIERR_INPUTLOST:
      case DIERR_NOTACQUIRED:
            dinKeyboard->Acquire();
            break;

      case E_PENDING:
            break;
      case DIERR_INVALIDPARAM:
      case DIERR_NOTINITIALIZED:
      default:
            // error has occured
            return(-1);
   }
   rc = dinKeyboard->GetDeviceState(256,keys);
}
```

```
if (key[DIK_LEFT] & 0x80)  rocket->fire();
```


Some of the Main keys

- **DIK_0 -9** On main keyboard
- **DIK_A-Z** a-z characters key
- **DIK_BACK** backspace
- **DIK_CAPITAL** caps lock
- **DIK_DELETE** delete
- **DIK_DOWN** down arrow - main keyboard
- **DIK_UP** up arrow
- **DIK_LEFT** left arrow
- **DIK_RIGHT** right arrow
- **DIK_END** end key
- **DIK_EQUALS** On main keyboard
- **DIK_ESCAPE** escape key
- **DIK_F1 -F15** F keys 0 -15
- **DIK_TAB**
- **DIK_SPACE** spacebar
- **DIK_LCONTROL** left control
- **DIK_RCONTROL** right control
- **DIK_RETURN** enter - main keyboard
- **DIK_NEXT** page down
- **DIK_PRIOR** page up
- **DIK_HOME**
- **DIK_INSERT**
- **DIK_LBRACKET** Left square bracket [
- **DIK_RBRACKET** Right square bracket]
- **DIK_LMENU** left ALT
- **DIK_RMENU** right alt
- **DIK_LSHIFT** Left SHIFT
- **DIK_RSHIFT** right shift
- **DIK_MINUS** On main keyboard
- **DIK_NUMPAD0 - NUMPAD1**
- **DIK_NUMPADENTER**
- **DIK_PAUSE**
- **DIK_PERIOD** On main keyboard
- **DIK_LWIN** Left Windows logo key
- **DIK_RWIN** Right Windows logo key
- **DIK_PLAYPAUSE**
- **DIK_VOLUMEDOWN**
- **DIK_VOLUMEUP**
- **DIK_MUTE**

Other keys

- **DIK_NUMPAD0 - NUMPAD1**
- **DIK_NUMPADENTER**
- **DIK_PAUSE**
- **DIK_PERIOD** On main keyboard
- **DIK_PLAYPAUSE**
- **DIK_POWER**
- **DIK_PRIOR** PAGE UP
- **DIK_RBRACKET** Right square bracket]
- **DIK_RCONTROL** Right CTRL
- **DIK_RETURN** ENTER on main keyboard
- **DIK_RIGHT** RIGHT ARROW
- **DIK_RMENU** Right ALT
- **DIK_RSHIFT** Right SHIFT
- **DIK_RWIN** Right Windows logo key
- **DIK_SCROLL** SCROLL LOCK
- **DIK_SEMICOLON**
- **DIK_SLASH** Forward slash (/) on main keyboard
- **DIK_SLEEP**
- **DIK_SPACE** SPACEBAR
- **DIK_SUBTRACT** MINUS SIGN (-) on numeric keypad
- **DIK_SYSRQ**
- **DIK_TAB**
- **DIK_UP** UP ARROW
- **DIK_VOLUMEDOWN**
- **DIK_VOLUMEUP**
- **DIK_WAKE**
- **DIK_WEBBACK**
- **DIK_WEBFAVORITES**
  - Displays the Microsoft Internet Explorer Favorites list, the Windows Favorites folder, or the Netscape Bookmarks list.
- **DIK_WEBFORWARD**
- **DIK_WEBHOME**
- **DIK_WEBREFRESH**
- **DIK_WEBSEARCH**
- **DIK_WEBSTOP**

Input Class

- Class that handles the require inputs
  - Can be reused
  - Makes the code "cleaner"
    - More complex? – needs to maintain more code
    - Can be tailored/modified/debugged only once
  - Can be instantiated more than once (needed?)

Input Class

```
Class gameInput {
Public:
// initialize the mouse and keyboard
gameInput(long mouseCoopertive,
          long keyboardCoopertive);

Release the input devices
~gameInput();

// get the state
int poll()

// key was pressed
int keyPressed(char key)

// key was released
int keyReleased(char key)

// change in position vector
D3DXVECTOR3 mouseDVector()
```

```
// change in position vector
D3DXVECTOR3 mouseDVector()

// change in x
int  mouseDX();

// change in y
int  mouseDX();

// change in z
int  mouseDX();

Private:
LPDIRECTINPUT8 din;

LPDIRECTINPUTDEVICE8 dinKeyboard
char keys[256];

IDirectInput8 * dinMouse;
DIMOUSESTATE  mouseState;}
```


Direct3D Initialization

- DirectX is a COM object (Component Object Model)
  - Can be viewed as a set of libraries (interfaces)
    - Direct3D – 3D graphics
    - DirectInput – for input (mouse, keyboard, joystick)
    - DirectSound – sound
    - + Other libraries

DirectX Initialization

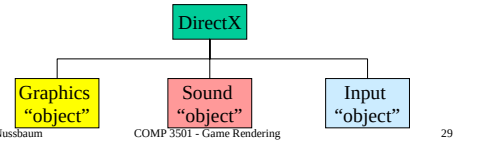
Acquire a handle to the DirectX COM object

The system checks what capabilities the HW can support

```
IDirect3D9 *d3d;
d3d = direct3DCreate(D3D_SDK_VERSION);
```

D3D_SDK_VERSION is defined in d3d9.h
#include <d3d9.h>

Note the DirectX object can also use the declaration
LPIDIREC3D9 d3d



Initialize the Graphics Device

Declare the graphics device

Direct3DDevice * d3dDev
Alternatively
LPDIRECT3DDEVICE9 d3dDevice

Create the Device

```
CreateDevice(
  UNIT Adapter,                          // which graphics card to use
  D3DDEVTYPE DeviceType,                 // how to execute draw commands
  HWND hWnd,                             // the window to be rendered
  DWORD BehaviourFlags,                  // device specifies behaviour
  D3DPRESENT_PARAMETERS * pPresentationParameters, , // window behaviour
  IDirect3DDvice9 **ppReturnedDeviceInterface;      // address of device handle
  )
```

Initialize the Graphics Device

```
IDirect3D9 *d3d=NULL;
D3DPRESENT_PARAMETERS d3dpp;
Direct3DDevice * d3dDev = NULL;

d3d = Direct3DCreate(D3D_SDK_VERSION);        get the DirectX interface

d3d->CreateDevice(                                    Create the device
  D3DDADAPTER_DEFAULT,      //  use the default adapter
  D3DDEVTYPE_HAL,           // use drawing on the graphics card
  hWnd,                     // the window to be rendered
  D3DCREATE_SOFTWARE_VERTEXPROCESSING,
  &d3dpp,                   // window specific behaviour
  &d3dDev;                  // address of device handle
   )
```

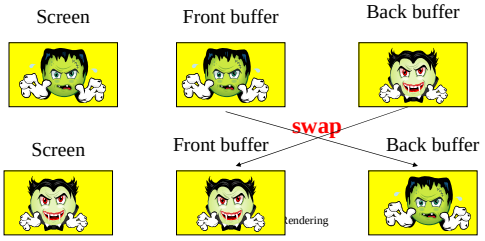

Double Buffers Motivation

- The HW uses a matrix (buffer) of pixels to contain the image to be displayed
  - Too slow
- The image on the screen is updated by the HW at a rate of 60-100HZ (60-100 times/sec.) by consulting the pixel in the buffer
- When a new item is to be rendered the pixel matrix is updated with the correct information.

- Issues
- Rendering is much faster than the screen refresh
- Leads to tearing - part of object being displayed
- May lead to screen flickering
  - Clear the screen and then render

Double Buffers

- Render into one buffer (back buffer) while refreshing the screen via another buffer (front buffer)



D3DPRESENT_PARAMETERS

- A structure containing window specific behaviour (may be modified by directx if number of backgound surfaces cannot be allocated)
- Important structure members
  - Windowed – display in a window or full screen
  - SwapEffect – how should the system swap between back and front buffers (D3DSWAPEFFECT_DISCARD)
  - DeviceWindow – the window to be rendered
  - BufferCount – how many back buffers to create (0 == 1)
  - BackBufferWidth, BackBufferHeigth – dimension of back buffer (if set to 0 then size if taken from the client area)

Example: D3DPRESENT_PARAMETERS

```
D3DPRESENT_PARAMETERS d3dpp;

ZeroMemory(&d3dpp, sizeof(d3dpp));

d3d.Windowed        = TRUE          // use a window
d3d.SwapEffect              = D3DSWAPEFFECT_DISCARD
d3d.hDeviceWindow           = hWnd;              // the app window
d3d.BackBufferFormat        = D3DFMT_X8R8G8B8;
d3d.BackBufferWidth         = SCREEN_WIDTH;
d3d.BackBufferHeight        = SCREEN_HEIGHT;
d3d.BackBufferCount         = 1;
d3d.MultiSampleType         = D3DMULTISAMPLE_NONE;
```

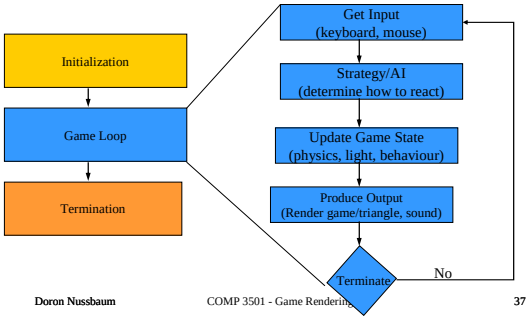
Note - There are more parameters that can be specified (see documentation)

Cleaning Up

- So far we ask the system for recourses
  - Direct3D handle
  - Graphics device handle
  - Memory for rendering buffers
- We must return the resources when done
  - Otherwise the object will run in the background (these are com objects).
- Executed in reverse order of creation

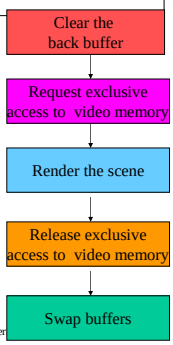
```
// Release the dev
      d3dDev->release()

// Release the Direct3D
      d3d->release();
```

Game Engine



Rendering

- Usually five steps
  - d3d->clear()
  - d3d->BeginScene()
  - ...
  - Rendering commands
  - ...
  - d3d->EndScene()
  - d3d->Present()



Clear the Back Buffer - Clear()

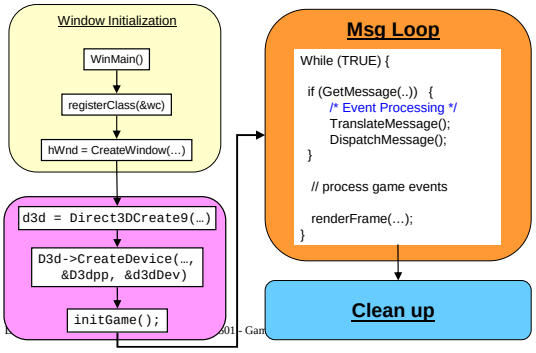
- Clears the back buffer
  - Can clear parts or all the screens
  - Sets the background colour
  - Sets some other buffers (for 3D visualization)

```
HRESULT Clear(
        DWORD Count,      // num rectangles
        D3DRECT * pRects,         // array of rectangles
        DWORD Flags,      // which surfaces to clear
        D3DCOLOR Color,   //background colour
        float Z,          // initial value of z buffer
        DWORD Stencil     // initial value of stencil buffer
);
```

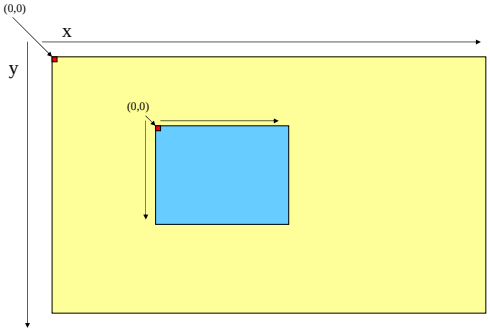

Rendering

```
void render_frame(void)
{
    d3dDevice->Clear(
        0, NULL,                    // clear all the client area
        D3DCLEAR_TARGET,            // clear the back buffer
        D3DCOLOR_XRGB(255, 0, 0), // red colour
        1.0f,                       // z buffer value (requires for 3D depth
        0);                         // stencil buffer initialization

    d3dDevice->BeginScene();    // lock the video memory
                                //(begins the 3D scene)

    // draw different objects of the scene into the back buffer

    d3dDevice->EndScene();      // release the video memory
                                // (ends the 3D scene)
    d3dDevice->Present(NULL, NULL, NULL, NULL);   // swap the buffers
    return;
}
```


Putting it all together



Screen Coordinates



What Can be Drawn?

Raster Data/Image
- Many small uniform cells
- Each covers a small area
  - uniform across area
- "No notion" of location
- Content has no explicit relationship to other cells
- 2D only

Vector Data
- Several types of data
- 1D, 2D, 3D or higher
- Can cover a larger area
- Area/region is not uniform
- Topological relationships
  - Adjacent
  - Near
  - Contained

Sprites

- Objects which are rendered as 2D images
  - Represented by a single 2D point and orientation
- "Flat images"
- Can be placed anywhere on the screen
- May be combined to give an illusion of 3D
- Limited direct usage in 3D
  - Provide guidance to player
    - Player data (e.g., health, score)
    - Controls

Handling and Drawing Sprites

- Create a sprite object manager
- Load the image
- Determine the location
- Draw one or more sprites

Using Sprites

- Create a sprite handler
- Load the image
- Determine the location
- Rendering the sprite
  - Displaying the required part of the image
- Releasing the sprite resource
  - Releasing the image

Sprite Management in DirectX

- Sprite rendering is managed by a com object (another resource)
  - D3DXSprite interface
- Obtaining the interface (com object)

```
LPD3DXSPRITE d3dSprite;  // a pointer to the resource
D3DXCreateSprite(d3dDevice, &d3dSprite);
```


Loading the Sprite Image

- Various ways to load the sprite image
  - From file, from memory
- Usually a sprite is loaded from a file
- Images can be of many types
  - .bmp, .dds, .jpg, .png…

D3DXCreateTextureFromFile(device, file name, image);

```
LPDIRECT3DTEXTURE mySprite;
D3DXCreateTextureFromFile(device, file name, sprite image);
D3DXCreateTextureFromFile(d3dDevice, L"car.bmp",&mySprite);
```

Rendering a Sprite

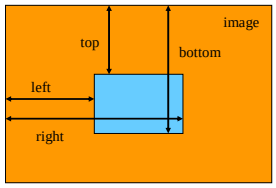
- "Placing" a binary image on the screen
  - Which image
  - Where to draw it
  - Which section
  - Intensity of the image

```
HRESULT Draw(
pTexture,       // the loaded image
RECT pSrcRect,
D3DVECTOR3 pCentre,
D3DVECTOR3 pPosition,
D3DCOLOR colour)
```


pSrcRect

```
HRESULT Draw(
pTexture,       // the loaded image
RECT pSrcRect,
D3DVECTOR3 pCentre,
D3DVECTOR3 pPosition,
D3DCOLOR colour)
```

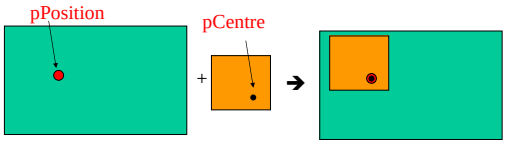
- Describes what portion of the loaded image is to be rendered
- A pointer to a RECT structure
  - 4 fields – top, left, bottom, right
  - Each field describes an edge of the rectangle



pPosition

pPosition
- Indicates the position of the image on the screen.
- Used in conjunction with the pCentre (pRefPoint)
- Indicates where the reference point should be on the screen.
  - pRefPoint and pPosition coincide
- The z-coordinate indicates the depth of the image can be used to order images of a scene

pCentre/ pRefPoint
- Defines a ref point on the image to be used to position the image on the screen
- A 3D vector where the x,y determines the reference point
- Top left corner of the rendered image has the coordinates (0,0)

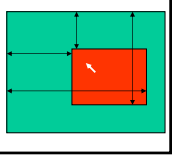


Colour

- The color indicates the intensity of the colours to be displayed.
- Creates side effects on the image presentation
  - Crazy colours
  - Eliminates some parts of the image
  - Fog effect
- Examples:
  - D3DCOLOR_XRGB(255,255,255) – full intensity
  - D3DCOLOR_XRGB(127,127,127) – 50% intensity
  - D3DCOLOR_XRGB(255,0,0) – show only red at 100%

Absolute coordinates

- Use cursor function
  - POINT pnt; // a point structure
  - GetCusorPos(&pt)
  - The function returns the absolute cursor position with respect to the screen and not the window
- The point structure
  - LONG *x, y;*
- Obtaining the absolute value within a window
  - Get the window rectangle dimension
    - Given with respect to the full screen (similar to sprite location)
    - Use function GetWindowRect(hWnd, &winRec);
    - hWnd – window handle, winRec a rectangle
  - Compute the cursor position since both are given with respect to the full screen
  - Note one must compensate for the border size
- One can also use the window mouse move event
  - Note only within the window



Putting it all together

- Rendering sprites is done via a batch process
  - Sprites are assembled together and then rendered
  - Allows manipulation between the sprites (e.g., depth)

```
d3dDevice->beginScene();
d3dSprite->Begin(NULL);

// draw the sprites
// all these sprites will be managed concurrently (batch)

d3dSprite->End();
d3dDevice->endScene();
d3dDevice->present();
```

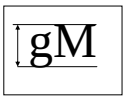
Depth

- Depth is handled by setting up the flags for the batch processing
- d3dSprite->Begin(D3DXSPRITE_SORT_TEXTURE | D3DXSPRITE_SORT_DEPTH_BACKTOFRONT)

Drawing sprites example

Drawing Text

- Adding text is simple
- Create a font
  - ***Courier, Bold, Italic, 28 pnt size***
  - ***Times New Roman, Bold, Italic, 28 pnt size***
- Render/Draw the text on the screen
- Font size
  - Measured from the bottom of g to top of M



Creating a font

```
HRESULT D3DXCreateFont(
    LPDIRECT3DDEVICE9 pDevice,      // the d3d device
    INT Height,                     // the height of the font
    UINT Width,                     // the width of the font (use 0 – don't care)
    UINT Weight,                    // the weight of the font value 0-1000
    UINT MipLevels,                 // number of MIPS levels
    BOOL Italic,                    // is the font italic?
    DWORD CharSet,                  // which character set to use
                                    // use DEFAULT_CHARSET (english)
    DWORD OutputPrecision,          // which similar font to use if font unknown
                                    // use OUT_DEFAULT_PRECIS (known font)
    DWORD Quality,                  // the output quality (different flags)
                                    // use DEFAULT_QUALITY (don't care)
    DWORD PitchAndFamily,           // # characters per inch use DEFAULT_PITCH
    LPCTSTR pFacename,              // the name of the font
    LPD3DXFONT* ppFont);            // a handle to the created font
```


Predefined font Weights

- FW_DONTCARE 0
- FW_THIN 100
- FW_ULTRALIGHT 200
- FW_LIGHT 300
- FW_REGULAR 400
- FW_MEDIUM 500
- FW_DEMIBOLD 600
- FW_BOLD 700
- FW_ULTRABOLD 800
- FW_BLACK 900

Create Example

```
LPD3DXFONT fontCourier;     // courier font object

D3DXCreateFont(
d3dDevice,          // the Direct3D Device
26,                 // height of font
0,                  // use the default width
FW_NORMAL,          // normal font weight
1,                  // no Mipmap
TRUE,               // italic
DEFAULT_CHARSET,    // default character set
OUT_DEFAULT_PRECIS, // default precision
DEFAULT_QUALITY,    // default quality
DEFAULT_PITCH || FF_DONTCARE,   // more defaults...
L"Courier",         // typeface "Courier"
&fontCourier);
```

Draw Text

- Similar to sprites

```
int DrawText(
LPD3DXSPRITE pSprite,     // the sprite object
LPCTSTR pString,          // the text string
INT Count,                // the number of characters to draw
                          // use -1 for '\0' terminated strings
LPRECT pRect,             // rectangle containing the text
DWORD Format,             // how to position the text within the rectangle
D3DCOLOR Color);          // text colour
```


Format Flags

- DT_SINGLELINE - Displays the text on a single line (ignores line breaks or <CR>)
- DT_CALCRECT – modifies the height and width of the rectangle (text dependent)
- DT_CENTER – align to centre.
- DT_LEFT – left alignment
- DT_RIGHT – right alignment
- DT_TOP – top justify
- DT_BOTTOM – bottom justify (only works with DT_SINGLELINE)
- DT_VCENTER – vertical centre justify (only works on single lines)
- DT_WORDBREAK – breaks lines between words when text is too long


```
RECT textBox(50,50,500,100);

// draw the game title "Asteroids the first game"

fontCourier->DrawTextA(
NULL,          // if not defined the system fetches it
"My Game!!",// the string
 -1,          // '\0' terminated string
 &textBox,          // the rectangle
 DT_CENTER | DT_VCENTER,  // formating
 D3DCOLOR_ARGB(255, 255, 0, 0));
```


The Game Loop

- What should be in the game loop?
- How does it relate to the OS (windows)?

Msg Loop

```
While (TRUE) {

   if (GetMessage(..))   {
    /* Event Processing */
    TranslateMessage();
    DispatchMessage();

   } else {

    // process game events
    renderFrame(…);
   }
}
```


The Game Loop

- Are there issues with the game loop?
- Yes,
  - Game may not proceed
  - Not enough time is given to the game

Msg Loop

```
While (TRUE) {

   if (GetMessage(..))   {
    /* Event Processing */
    TranslateMessage();
    DispatchMessage();

   } else {

    // process game events
    renderFrame(…);
   }
}
```


The Game Loop

- Replace GetMessage with PeekMessages
  - Use remove message
- Ensure that control is given to the game
  - Do not consume resources
  - Use appropriate time slice

Msg Loop

```
While (TRUE) {

   if (GetMessage(..))   {
    /* Event Processing */
    TranslateMessage();
    DispatchMessage();

   } else {

    // process game events
    renderFrame(…);
   }
}
```

Summary

- Short review of gaming
- Review of Direct3D
  - Graphics initialization
  - Input devices - mouse and keyboard
  - Output – fonts and sprites

What's next?

- Graphics Rendering pipeline
- 3D Geometry
  - Vertices
  - Quads
  - transofrmations
- Direct3D
  - Vertex buffer
  - Index buffer
  - Primitives
  - …