**ביבליוגרפיה**

1. מסביר על הפעולה WinMain

http://msdn.microsoft.com/en-us/library/windows/desktop/ms633559(v=vs.85).aspx

1. מה זה HANDLE

<http://stackoverflow.com/questions/1303123/what-is-a-handle-in-c>

1. איך הפעולות GetMessage ו-PeekMessages עובדות:

<http://msdn.microsoft.com/en-us/library/windows/desktop/ms644936(v=vs.85).aspx>

<http://www.directxtutorial.com/tutorial9/A-Win32/dx9A4.aspx>

1. איך לטעון תמונה (טקסטורה) לחלון באמצעות DirectX 9:

<http://cboard.cprogramming.com/game-programming/120768-directx-9-2d.html>

1. מה זה FVF:

<http://msdn.microsoft.com/en-us/library/windows/desktop/bb172559(v=vs.85).aspx>

1. הפעולה CreateVertexBuffer:

<http://msdn.microsoft.com/en-us/library/windows/desktop/bb174364(v=vs.85).aspx>

1. ההבדל בין ההודעות WM\_QUIT, WM\_CLOSE ו – WM\_DESTROY:

<http://stackoverflow.com/questions/3155782/what-is-the-difference-between-wm-quit-wm-close-and-wm-destroy-in-a-windows-pr>

1. איך ליצור פונט:

<http://msdn.microsoft.com/en-us/library/windows/desktop/bb172773(v=vs.85).aspx>

1. איך לרנדר טקסט:

<http://www.toymaker.info/Games/html/text.html>

1. איך לשנות את צבעי הטקסטורה:

<http://stackoverflow.com/questions/1988387/turning-a-directx-sprite-solid-white>

**מידע**

מחלקת Window

**1) איך פותחים חלון ב-C++ :**

Windows GUI tend to follow a common trend:  
Step:  
1. Register a Window Class;  
2. Create a (main) window of that class.  
3. Enter the message loop.  
  
You also provide a window procedure to handle message for the window you created.  
  
The message loop generally looks like something like this:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | {  MSG msg;  while (GetMessage(&msg, NULL, 0, 0) > 0) {  TranslateMessage(&msg);  DispatchMessage(&msg);  }  return msg.wParam;  } |

This message loop gets messages from your window queue.  
Now this look like all windows messages for your window will go through this queue - but that is not true.  
Most messages that windows generate for your application are sent directly to your Window Procedure.  
  
Some message types that go through the queue and therefore are picked up and despatched by this loop are:  
WM\_PAINT  
keyboard related messages.  
mouse related messages  
  
Now keyboard related messages arrive in the queue as Virtual key codes - the prupose of the**TranslateMessage()** function is to translate keyboard messages into WM\_CHAR messages.  
  
The **DespatchMessage()** function sends the message off to the window procedure.

**2) הפעולה ourWindowProcedure חייבת להיות סטטית כשהיא בתוך מחלקה מכיוון:**

תגובות:

"You can't use a non-static member function as a window procedure. If you declare WndProcedure as static it should compile. A non-member function would work as well.

Non-static member functions have a different signature than static members. This is because they receive an implicit this parameter in addition to the explicitly defined parameters."

"That's because your WndProcedure function must be either a global function or a **static** member function."

"replace

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

by

static LRESULT CALLBACK WndProcedure(HWND, UINT, WPARAM, LPARAM);

The this pointer is a hidden parameter in your function call and by declaring it static the this pointer is not a parameter anymore and the signature of the two functions match."

**3) ההבל בין הפעולה PeekMessage לפעולה GetMessage:**

**GetMessage** will check the message queue for message, if there aren't any messages in the queue it will *block*. Blocking, in this case, would mean that GetMessage will wait for a valid message to pop up into the message queue.

If the function retrieves a message other than [**WM\_QUIT**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms632641(v=vs.85).aspx), the return value is nonzero.

If the function retrieves the [**WM\_QUIT**](http://msdn.microsoft.com/en-us/library/windows/desktop/ms632641(v=vs.85).aspx) message, the return value is zero.

If there is an error, the return value is -1. For example, the function fails if hWnd is an invalid window handle or lpMsg is an invalid pointer. To get extended error information, call **[GetLastError](http://msdn.microsoft.com/en-us/library/windows/desktop/ms679360(v=vs.85).aspx)**.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | //GetMessage loop example.  while (GetMessage (&uMsg, NULL, 0, 0) > 0)  {  TranslateMessage (&uMsg);  DispatchMessage (&uMsg);  } |

**PeekMessage** will check the message queue and return the first message the first message that enters the queue. If a message is available, the return value is nonzero.

If no messages are available in the queue, the return value is zero.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | //PeekMessage loop example  while (WM\_QUIT != uMsg.message)  {  while (PeekMessage (&uMsg, NULL, 0, 0, PM\_REMOVE) > 0) //Or use an if statement  {  TranslateMessage (&uMsg);  DispatchMessage (&uMsg);  }  } |

GetMessage = wait for message.  
PeekMessage = return the first message, or return nothing if there are no messages.

**4) כיצד לטעון לחלון תמונה (טקסטורה) דו-מימדית לעולם תלת-מימדי באמצעות DirectX 9:**

טכניקת טעינת התמונה שאני הולך להציג לא מצריכה מצלמה והיא נעשית בצורה כזו שהתמונה ממוקמת בצורה אבסולוטית בחלון.

לטעינת טקטורה מספר שלבים:

4) 1. טעינת הטקטורה בשלמותה מן הקובץ ויצירת pointer שמצביע למקום בזיכרון בו אליו נטענה הטקטורה השלמה:

IDirect3DTexture9 \*g\_pTexture=NULL; // Pointer To The Texture

D3DXCreateTextureFromFile(refDevice.p\_dx\_Device, //Direct3D Device

"pacman2.png", //File Name

&g\_pTexture); //Texture handle

4) 2. יצירת מבנה נתונים (Struct) שיכיל בתוכו מידע על תצוגת הטקטורה בחלון (גודל, מיקום וכו'):

// ABSOLUTE Vertex Struct

struct CUSTOMFLATVERTEX

{

float x, y, z, rhw; // The transformed(screen space) position for the vertex.

float tu, tv; // Texture coordinates

DWORD color;

};

#define CUSTOMFLATFVF (D3DFVF\_XYZRHW | D3DFVF\_TEX1)

כאשר רוצים לטעון טקסטורה למרחב דו-מימדי אנו צריכים שה-Vertex-ים יכילו את הנתונים:

- מיקום במרחב (x,y,z), מכיוון שאנו טוענים תמונה לעולם דו מימדי ציר ה-Z מאופס.

- RHW – Reciprocal Homogenous W  הערך שמכיל ה-RHW הוא: העומק של ה-VERTEX / 1.

<http://stackoverflow.com/questions/9954152/what-is-rhw-in-directx-vertex>

- צבע (DWORD Color)

- Texture Coordinate – הערכים tu ו-tv.

לאחר מכן אנו צריכים להגדיר FVF (Flexible Vertex Format) שהוא למעשה גרסה ישנה של ה-VertexDecleratoin. ה-FVF מתאר לכרטיס מסך את מבנה הנתונים שיצרנו (סוג הנתונים שמכיל כל VERTEX ב- Vertex Buffer שנשלח אליו) או במילים פשוטות יותר, הפורמט של ה-Vertices (שם נוסף שניתן לפורמטים השונים הוא Flags). במקרה שלנו הפורמטים של ה- Verticesהם:

**D3DFVF\_XYZRHW** **–** הוא Vertex Format כאשר Vertex-ים מפורמט זה מכילים מידע על מיקומם (X,Y,Z). –Vertexים מפורמט זה עברו טרנספורמציה. הפורמט משמש למרחב הדו-מימדי ולכן לא יכול להיות משומש יחד עם הפורמט **D3DFVF\_XYZ** (המכיל מידע על מיקום Vertex, שלא עבר טרנספורמציה, בעולם תלת-מימדי, ולכן לא יכול להיות משולב עם פורמט לעולם דו-מימדי) או הפורמט **D3DFVF\_NORMAL** (המכיל מידע על הנורמל של ה-VERTEX).

**D3DFVF\_TEX1 –** הוא Vertex Format הנע בין **D3DFVF\_TEX0-D3DFVF\_TEX8**. Vertex-ים מפורמט זה מכילים קבוצות שלTexture Coordinate. Texture Coordinate מתייחס לערכים tu ו-tv במבנה נתונים שיצרנו.

4) 3. הגדרת מערך של 4 משתנים ממבנה הנתונים שיצרנו (עבור כל פינה בטקסטורה):

CUSTOMFLATVERTEX vertices[] =

{

{0.0f, 50.0f, 0.0f, 1.0f, 0.0f, 1.0f, D3DCOLOR\_XRGB(255, 255, 255)},

{0.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, D3DCOLOR\_XRGB(255, 255, 255)},

{50.0f, 50.0f, 0.0f, 1.0f, 1.0f, 1.0f, D3DCOLOR\_XRGB(255, 255, 255)},

{50.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f, D3DCOLOR\_XRGB(255, 255, 255)}

};

4) 4. הגדרות מנגנון זיכרון זמני (Buffer) שבתוכו המערך שיצרנו:

קצת על **VertexBuffers**- VertexBuffers הן חבילות המכילות מידע (**VertexData**) על Vertex-ים (מיקום, צבע, נורמל וכו') הממוקמות בנקודת בזיכרון מהם יכולים יישומים, אותם אנו בוחרים, לשאוב את המידע באופן ישיר. ראשית, מייצרים מבנה נתונים (Struct) שיאפיין כל Vertex, לאחר מכן יוצרים מערך Vertex-ים מסוג מבנה הנתונים שיצרנו, לאחר מכן מייצרים את החבילה (VertexBuffer) ולבסוף מעתיקים את המידע ל-VertexBuffer (מעתיקים את המערך אל ה-Buffer). השימוש ב-VertexBuffers מייעל את פעולת היישום שכן הוא לא צריך לגשת אל הנקודה בזיכרון בה נוצר המידע, אלא יכול לגשת אל ה-VertexBuffer שמכיל את המידע ו נמצא בנקודה בזיכרון אליה יכול היישום לגשת באופן ישיר.

ב- DirectX Device ישנה פעולה שכל מטרתה יצירת VertexBuffer – **"pointer to directx device"->CreateVertexBuffer**.

הפעולה מקבלת:

* **UINT Length** **-** גודל ה-VertexBuffer בזיכרון = מספר ה-Vertex-ים כפול גודל מבנה הנתונים שיצרנו.
* **DWORD Usage –** ה-VertexBuffer מכיל בתוכו את ה-Vertex-ים שיצרנו וכעת הוא נשלח לחומרה (כרטיס מסך) או לתוכנה המדמה את פעולות ה-DirectX (Reference Device). ה-VertexBuffer יישלח לתוכנה רק במקרה שכרטיס המסך לא חזק מספיק לעיבוד ה-Vertex-ים (התוכנה איטית לעומת כרטיס המסך). הפרמטר **Usage** מורה ל-VertexBuffer אם לעבד את ה-Vertex-ים בחומרה (במקרה הזה הערך חייב להיות 0) או בתוכנה (במקרה הזה הערך חייב להיות או 0 או D3DUSAGE\_SOFTWAREPROCESSING).
* **DWORD FVF -**  הפורמט של ה-Vertex-ים.
* **D3DPOOl Pool –** D3DPOOL הוא משתנה מסוג enum. ה-D3DPOOL מכיל רשימת שמות, כאשר כל אחד מהם מגדיר שיטת אחסון שונה של ה- VertexBuffer(משמעות ה"אחסון" הוא מיקום בזיכרון בו ישמש ה-VertexBuffer כמקור לשאיבת מידע). אנו משתמשים בשיטת האחסון ה נקרא **D3DPOOL\_MANAGED**. באמצעות שיטה זו ה-VertexBuffer מועתק אוטומטית לזיכרון הנגיש ל-DirectX Device.

בנוסף ניתן לנעול (באמצעות הפעולה **Lock**) את המידע שב-VertexBuffer בשיטת האחסון הזו.

* **IDirect3DVertexBuffer9 \*\*ppVertexBuffer –**  הכתובת בזיכרון של המצביע ל-VertexBuffer.
* **Handle\* pSharedHandle –** הערך תמיד יהיה NULL.

**HRESULT -**  אם הפעולה מצליחה יוחזר **D3D\_OK**.

4) 5. העתקת המידע אל ה-VertexBuffer:

להעתקת המידע אל ה-VertexBuffer שלושה שלבים:

* נעילת קטע מסוים ב-VertexBuffer, או את כל ה-VertexBuffer, ע"מ למלא אותו במידע/ לקרוא ממנו מידע. הנעילה נעשית באמצעות הפעולה **Lock**. הפעולה מקבלת ארבעה פרמטרים: - **UINT Offset**- הנקודה ב-VertexData ממנה תתחיל הנעילה (ע"מ לנעול את כל המידע ב-VertexBuffer יוכנס הערך 0 לפרמטר הזה ולפרמטר הבא). - **UINT SizeToLock** – גודל ה-VertexData שברצוננו לנעול. - **VOID \*\*ppbData** - הפרמטר הוא כתובת של מצביע המצביע ל-VertexData שאנו ועומדים לנעול. - **DWORD flags –** הפרמטר מורה למערך כיצד לנעול את ה-VertexData, ערך ה-Default הוא 0. ניתן לבצע נעילות מסוגי ReadOnly או WriteOnly.

// pointer to the soon-to-be-

// locked VertexData

VOID\* pVoid;

// locks a specific range of bytes

// in the vertex buffer in order

// to read the bytes or write to the

// bytes without them being accessed

// by other applications

pVertexBuffer->Lock(

0, // The offset byte to start lock from

// (if this parameter and the next parameter

// are set to 0, al the vertex buffer is locked)

0, // the amount of bytes to lock

(void\*\*)&pVoid,

0); // the type of lock (read only/write only etc...)

* העתקת המידע (המערך Vertices) אל ה-VertexBuffer באמצעות הפעולה **memcpy**:

memcpy(

pVoid, // Destination - pointer to the VertexData

vertices, // Sorce - The Vertex Data the is copied to the VertexBuffer

sizeof(vertices)); // Size - The Size of the Vertex Data

* שחרור הקטע הנעול ב-VertexData ע"מ לאפשר גישה ליישומים באמצעות הפעולה **Unlock**.

// Releases the locked bits

pVertexBuffer->Unlock();

4) 6. העברת כל המידע ל-DirectX וציור הטקסטורה:

//DRAW ABSOLUTE IMAGE

// Select Texture

refDevice.p\_dx\_Device->SetTexture( 0, g\_pTexture );

// select which vertex format we are using

refDevice.p\_dx\_Device->SetFVF(CUSTOMFLATFVF);

// select the vertex buffer to display

refDevice.p\_dx\_Device->SetStreamSource(0, v\_buffer, 0, sizeof(CUSTOMFLATVERTEX));

// copy the vertex buffer to the back buffer

refDevice.p\_dx\_Device->DrawPrimitive(D3DPT\_TRIANGLESTRIP, 0, 2);

**5) כיצד לסגור חלון:**

כאשר המשתמש לוחץ על כפתור ה-**X** בצד הימני העליון של החלון (או לוחץ Alt+F4, או בוחר ב-Close מתפריט בחלון), מתקבלת הודעה ב-**MessageQueue** בשם: **WM\_CLOSE**. ההודעה נקלטת ב-**MessageLoop** ע"י הפעולה **()PeekMessages** ומשם מועברת ל-**DefWindowProc**. ב-DefWindowProc אנו יכולים לבחור כיצד להגיב להודעה שנקלטה ((WM\_CLOSE: להתעלם ממנה ולהשאיר את החלון פתוח או להשמיד את החלון.

במקרה שאנו בוחרים להשמיד את החלון נשלחות הודעות ב-MessageQueue בשם  **WM\_DESTROY**ו- **WM\_NCDESTROY**. ההודעות נקלטות ע"י הפעולה PeekMessages ונשלחת ל-defWindowProc. ההודעה **WM\_NCDESTROY** נשלחת כאשר החלונות הפנימיים לחלון הראשי (**Child Windows** אם יש) מושמדים. ההודעה **WM\_DESTROY** נשלחת כאשר כל החלונות הפנימיים הושמדו וכעת החלון הראשי בתהליך השמדה. כאשר ההודעה WM\_DESTROY נשלחת אין באפשרותינו לבטל את ההשמדה אלא רק לשחרר מאגרי זיכרון אחרונים אשר היו תפוסים ע"י היישום שלנו וניקויים אחרונים נוספים.

לאחר שהחלון נסגר (WM\_CLOSE) ונהרס (WM\_DESTROY) לולאת ההודעת האינסופית (**MessageLoop**) עדיין ממשיכה לרוץ. הדרך הנכונה לעצור את לולאת ההודעות כאשר החלון נסגר היא באמצעות הפעולה **PostQuitMessage()**. הפעולה נקראת במקרה שאנו רוצים לסגור את היישום סופית (כולל עצירת לולאת ההודעות). הפעולה שולחת הודעה ב-MessagesQueue בשם **WM\_QUIT**. ההודעה WM\_QUIT מעידה על כך שהיישום רוצה להיסגר, התהליך (Thread) רוצה להתפנות, אם לא תשלח, לולאת ההודעות תמשיך לרוץ. ההודעה לא קשורה לאף חלון ביישום ולכן היא אינה נשלחת ל-DefWindowProc.

אז אחרי שהסברתי את התיאוריה, אציג את הקוד.

ראשית ה-DefWindowProc קולט את ההודעה WM\_CLOSE:

switch(uint\_Message)

{

case WM\_CLOSE:

{

PostQuitMessage(0);

DestroyWindow(hWnd);

break;

}

break;

}

במקרה שנקלטת ההודעה מתבצעות שתי פעולות: הראשונה PeekMessages אשר שולחת הודעת WM\_QUIT ל-MessageQueue ומודיע למעשה שהיישום רוצה להיסגר. והשניה DestroyWindow אשר מקבלת את ה-HANDLE של החלון אותו אנו רוצים להשמיד.

עכשיו כל שנותר הוא לקלוט את הודעת ה-WM\_QUIT בלולאת ההודעות (כזכור היא אינה מגיעה ל-DefWindowProc).

if(PeekMessage(&msg,NULL,0,0,PM\_REMOVE))

{

if(msg.message == WM\_QUIT)

break;

// Translate keyboard messages into WM\_CHAR messages.

TranslateMessage(&msg);

// Sends the messages to the window procedure

DispatchMessage(&msg);

}

כך אנו קולטים את ההודעה WM\_QUIT, ולאחר שנקלטה יוצאים מן הלולאה האינסופית באמצעות המילה השמורה **Break**.

**6) כיצד לטעון תמונה (טקסטורה) באמצעות Sprite:**

השימוש ב-Sprite ע"מ לטעון תמונה הוא הנפוץ כאשר רוצים ליצור משחק דו מימדי.