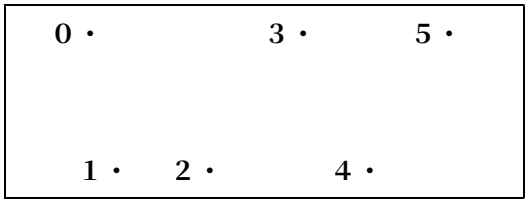
オブジェクト指向と ゲームプログラミング

▶ DirectX Graphics 3D編 － 第１回　プリミティブと頂点バッファ ◀

プリミティブ

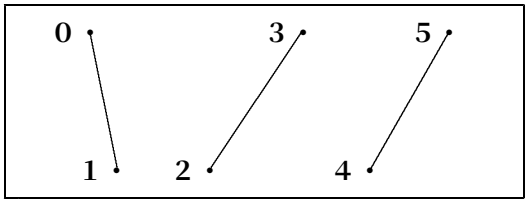
プリミティブとは、3Dオブジェクトを形成する頂点の集まりのことです。複数の頂点をなんらかの方法で繋げることにより、プリミティブが構成されます。プリミティブを複数集めることにより、3Dオブジェクトが形成されます。
DirectX Graphicsでは、プリミティブを以下の６つの方法で描画することができます。

・点リスト



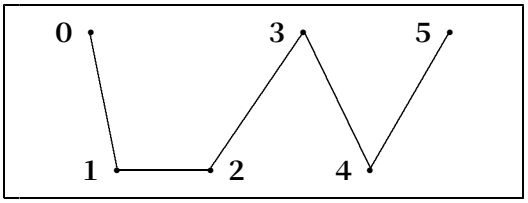
頂点を点で描画します。

・ラインリスト



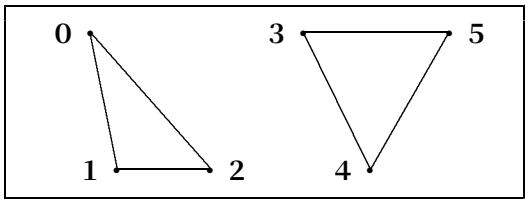
２つの頂点を別個の線分のリストとして描画します。

・ラインストリップ



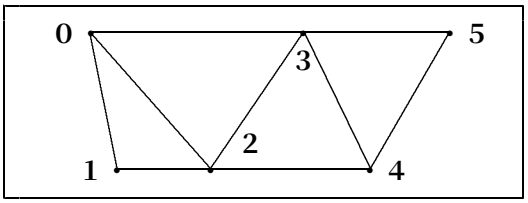
頂点を連続した線分のリストとして描画します。

・三角形リスト



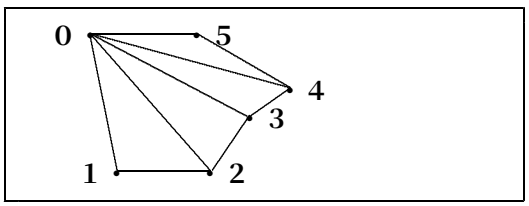
３つの頂点を別個の三角形のリストとして描画します。

・三角形ストリップ



頂点を連続した三角形として描画します。

・三角形ファン



頂点を三角形の扇型として描画します。

柔軟な頂点フォーマット

頂点を描画するには、頂点の座標や色といった情報(頂点データ)が必要になります。DirectX Graphicsでは、プログラマが頂点データのフォーマットを定義することになっています。
任意のフォーマットを定義できることから、柔軟な頂点フォーマット(FVF:Flexible Vertex Format)と呼びます。柔軟な頂点フォーマットに含めることのできるおもな情報は、以下のものです。

x，y，z座標　　法線ベクトル　　ディフューズ色　　スペキュラ色　　テクスチャのＵＶ値
RHW　　頂点ブレンディングの重み　　ポイントサイズ

これらを組み合わせて頂点データのフォーマットを定義します。ただし、RHWと法線ベクトルのように、同時に使用できないデータもあります。また、格納する順序は、以下のように決められています。

座標(float×３)
RHW(float)
頂点ブレンディングの重み(float×１～３)
頂点の法線ベクトル(float×３)
頂点のポイントサイズ(float×３)
ディフューズ色(D3DCOLOR)
スペキュラ色(D3DCOLOR)
テクスチャのＵＶ値１(float×１～４) ⋮ テクスチャのＵＶ値８(float×１～４)

頂点データは構造体で定義します。頂点データ構造体の配列を後述のIDirect3DDevice9::DrawPrimitive系のメソッドに渡せば、プリミティブが描画されます。なお、このままでは、どのようなフォーマットなのかDirectX Graphicsは認識できないので、事前にIDirect3DDevice9::SetFVFメソッドで知らせておきます。
SetFVFメソッドに渡す値は、以下のフラグの組み合わせになります。

フラグ	内　容
D3DFVF_XYZ	未座標変換の頂点座標(モデル座標系の頂点)
D3DFVF_XYZRHW	座標変換済みの頂点座標(スクリーン座標系の頂点)
D3DFVF_XYZB1～D3DFVF_XYZB5	頂点ブレンディングの重み
D3DFVF_NORMAL	頂点の法線ベクトル
D3DFVF_PSIZE	頂点のポイントサイズ
D3DFVF_DIFFUSE	頂点のディフューズ色
D3DFVF_SPECULAR	頂点のスペキュラ色
D3DFVF_TEX0～D3DFVF_TEX8	テクスチャのＵＶ値(D3DFVF_TEX0は指定無しと同じ) マルチテクスチャのため、１～８まで用意されている

よく使われるのが、「トランスフォーム済みライティング済みの頂点」「未トランスフォームライティング済みの頂点」「未トランスフォーム未ライティングの頂点」です。それぞれ、以下のように構造体とフラグを定義します。

・トランスフォーム済みライティング済みの頂点(色付きのスクリーン座標の頂点)
スクリーン座標と色を指定して頂点を描画することができます。画面上の４点を結んで四角形ポリゴンを作り、テクスチャを貼ることによって2Dキャラクタを描画することができます。

```
struct TLVERTEX {
```

```
    float      x, y, z, rhw;    // 座標変換済みの頂点座標
    D3DCOLOR   color;           // 頂点の色
    float      tu, tv;          // テクスチャのＵＶ値
};

// FVF設定(pD3DDeviceは、初期化済みのDirect3DDevice9オブジェクトのインタフェース)
pD3DDevice->SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1);
```

・未トランスフォームライティング済みの頂点(色付きのモデル座標の頂点)
　3Dオブジェクトの原点がもとになる座標系の頂点です。座標変換はDirectX Graphicsが行い、頂点の色は自ら計算するときに使用します。

```
struct LVERTEX {
    float      x, y, z; // 頂点座標
    D3DCOLOR   color;       // 頂点の色
    float      tu, tv;      // テクスチャのＵＶ値
};

// FVF設定
pD3DDevice->SetFVF(D3DFVF_XYZ | D3DFVF_DIFFUSE | D3DFVF_TEX1);
```

・未トランスフォーム未ライティングの頂点(法線方向ベクトルを持つモデル座標)
　3Dオブジェクトの原点がもとになる座標系の頂点です。座標変換も照明演算もDirectX Graphicsに行わせるときに使用します。頂点の色は、ライト、マテリアル、法線の向きによって計算されます。

```
struct VERTEX {
    float   x, y, z;    // 頂点座標
    float   nx, ny, nz; // 法線ベクトル
    float   tu, tv;     // テクスチャのＵＶ値
};

// FVF設定
pD3DDevice->SetFVF(D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1);
```

プリミティブの描画

　頂点データの配列など、プログラマが確保したメモリ領域に格納されたプリミティブの描画は、IDirect3DDevice9::DrawPrimitiveUPメソッドで行います。

IDirect3DDevice9::DrawPrimitiveUPメソッド

－説明－
　DrawPrimitiveUPメソッドは、プログラマが確保したメモリ領域(ユーザメモリポインタ)に格納された頂点データを、指定された方法でレンダリングします。

－書式－

```
HRESULT DrawPrimitiveUP(D3DPRIMITIVETYPE PrimitiveType, UINT PrimitiveCount,
                        const void* pVertexStreamZeroData, UINT VertexStreamZeroStride);
```

－パラメータ－
　１つ目の引数(PrimitiveType)は、レンダリングの方法です。以下の値のいずれかを指定します。

D3DPT_POINTLIST	点リスト
D3DPT_LINELIST	ラインリスト
D3DPT_LINESTRIP	ラインストリップ
D3DPT_TRIANGLELIST	三角形リスト
D3DPT_TRIANGLESTRIP	三角形ストリップ
D3DPT_TRIANGLEFAN	三角形ファン

　２つ目の引数(PrimitiveCount)は、レンダリングするプリミティブの数です。プリミティブの数とは、頂点の数ではなく、点、線、三角形をいくつレンダリングするのかということです。

　３つ目の引数(pVertexStreamZeroData)は、頂点データが格納された領域のアドレスを指定します。

　４つ目の引数(VertexStreamZeroStride)は、次の頂点データまでのバイト数を指定します。

－戻り値－
成功した場合はD3D_OK、それ以外はエラーコードを返します。

```
// 構造体定義
struct TLVERTEX {
    float   x, y, z, rhw;   // 座標変換済みの頂点座標
    DWORD   color;          // 頂点の色
    float   tu, tv;         // テクスチャのＵＶ値
} tlv[6];                   // ６つの頂点

// プリミティブ描画(三角形を２つ描画)
pD3DDevice->DrawPrimitiveUP(D3DPT_TRIANGLELIST, 2, tlv, sizeof(TLVERTEX));
```

頂点バッファ

頂点バッファ(Vertex Buffer)は、DirectX Graphicsが直接扱うことのできる、頂点データを格納する領域です。頂点バッファに格納された頂点は、DirectX Graphicsのすべての機能を使用してレンダリングすることができます。また、頂点バッファはビデオメモリ(VRAM)やAGPメモリに作成することができるので、システムメモリを用いる頂点データの配列に比べ、レンダリング速度の向上が望めます。
頂点バッファは、IDirect3DDevice9::CreateVertexBufferメソッドで生成します。

IDirect3DDevice9::CreateVertexBufferメソッド

－説明－
CreateVertexBufferメソッドは、頂点バッファを生成します。

－書式－

```
HRESULT CreateVertexBuffer(UINT Length, DWORD Usage, DWORD FVF, D3DPOOL Pool,
                           IDirect3DVertexBuffer9** ppVertexBuffer, HANDLE* pHandle);
```

－パラメータ－
１つ目の引数(Length)は、頂点バッファのサイズです。最低でも頂点データが１つ格納できる大きさにしなければなりませんが、頂点データの倍数である必要はありません。

２つ目の引数(Usage)は、０または頂点バッファの使用法フラグを指定します。おもに以下のフラグを組み合わせを指定します。

D3DUSAGE_DYNAMIC	動的なメモリを使用します。このフラグを指定しない場合、静的なメモリが使用されます。 書き込みを多く行うバッファの場合､このフラグを使用すると、レンダリングを行いながら書き込むことができるようになるので、書き込み効率が向上します。ただし、バッファはAGPメモリに作成されるため、レンダリング速度が若干低下します。
D3DUSAGE_WRITEONLY	書き込み専用バッファになります(レンダリングはできます)。このフラグを指定すると、書き込み効率が向上します。
D3DUSAGE_SOFTWAREPROCESSING	頂点処理をソフトウェアで行います。指定しない場合、ハードウェアで行われます。

３つ目の引数(FVF)は、バッファに格納する頂点データのFVFフラグです。

４つ目の引数(Pool)は、頂点バッファを配置する以下のメモリクラスフラグを指定します。

D3DPOOL_DEFAULT	頂点バッファを格納するのに最も適したメモリ(VRAMまたはAGPメモリ)に配置されます。このフラグを使って生成されたリソースは、IDirect3DDevice9::Resetメソッドを呼び出す前に、解放しなければなりません
D3DPOOL_MANAGED	DirectX Graphicsがアクセスできるメモリに頂点バッファを生成し、さらにシステムメモリにバックアップを作成して自動メモリ管理を行います。ほとんどの場合、このフラグを用います
D3DPOOL_SYSTEMMEM	システムメモリに生成されます
D3DPOOL_SCRATCH	システムメモリに生成されます｡このフラグを用いて生成されたリソースは、サイズやフォーマットの制限を受けません

５つ目の引数(ppVertexBuffer)は、生成される頂点バッファオブジェクトのインタフェースを受け取る変数のアドレスです。IDirect3DVertexBuffer9*型またはLPDIRECT3DVERTEXBUFFER9型の変数のアドレスを指定します。

６つ目の引数(pHandle)は、予約されておりNULLにしなければなりません。

－戻り値－
成功した場合はD3D_OK、それ以外はエラーコードを返します。

```
// 頂点バッファ生成(1024バイトの「色つきの座標変換済み頂点」を格納する頂点バッファ)
IDirect3DVertexBuffer9*   pVertexBuffer;
pD3DDevice->CreateVertexBuffer(1024, 0, D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1,
                               D3DPOOL_MANAGED, &pVertexBuffer, NULL);
```

課　　題

CDXGraphics9クラスに、頂点バッファを生成および解放する機能を追加しましょう。

(１)頂点バッファをカプセル化するクラスを作成します。

頂点バッファは、IDirect3DVertexBuffer9インタフェースをとおして操作を行います。このインタフェースは、頂点バッファの生成に失敗した場合、NULLを保持します。ほとんどの場合、頂点バッファが無いということは、3D処理が行えないということになり、プログラムを終了させます。しかし、場合によっては、実行を続けたいこともあります。

この場合、IDirect3DVertexBuffer9インタフェースがNULLの場合とそうでない場合が考えられます。NULLは具体的なインスタンスを参照しておらず、そもそも読むことも書くこともできない領域を指しているので、NULLを介してメソッドを呼び出すと、Windowsは例外を発生させてプログラムを強制終了します。例外が発生しないようにするためには、以下のように、その都度インタフェースがNULLかどうかを調べなければなりません。

```
// pVertexBufferは、Direct3DVertexBuffer9オブジェクトのインタフェース
if(pVertexBuffer != NULL) {
    // pVertexBufferを使った処理(頂点バッファへの書き込みなど)
}
```

しかし、プログラムによってはこのようなチェックが非常に多くなってしまい、コードが煩雑になってしまう危険性があります。そこで、デザインパターンのNull Object(Null Device)パターンを適用します。
Null Objectパターンは、インタフェースやポインタがインスタンスを参照していないことを表現するのに、NULLを用いるのではなく、明示的に「何もしないオブジェクト」(Null Object)を使うようにする方法です。そのNull Objectは、呼び出される可能性のあるメソッドはすべて持ちますが、すべて「何もしない」処理として実装します。このようなNull Objectを導入すると、NULLチェックを省くことができ、簡潔なコードになります。

Null Objectパターンを適用した頂点バッファクラスのヘッダファイル(VertexBuffer.hpp)を以下のように作成しましょう。

－VertexBuffer.hpp－

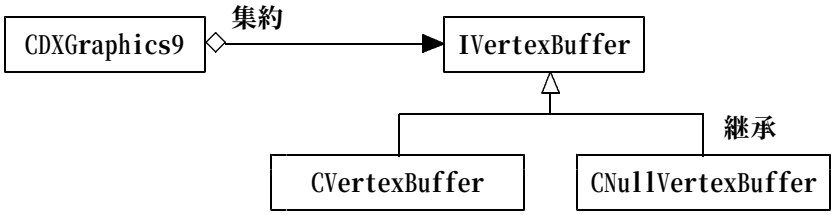
```
/*
===================================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
===================================================================================================

【対象ＯＳ】
   Microsoft Windows2000/XP

【コンパイラ】
   Microsoft Visual C++ 2005

【プログラム】
   VertexBuffer.hpp
              頂点バッファクラスヘッダ

【履歴】
   * Version    1.00       2005/03/dd  hh:mm:ss

===================================================================================================
*/
```

```
#pragma once

/*****************************************************************************************************/
/*                                          インクルードファイル                                     */
/*****************************************************************************************************/
#include <d3dx9.h>

/*****************************************************************************************************/
/*                                       頂点バッファインタフェース定義                              */
/*****************************************************************************************************/
class IVertexBuffer {
public:
    virtual ~IVertexBuffer() {}
};

/*****************************************************************************************************/
/*                                       頂点バッファクラス定義                                      */
/*****************************************************************************************************/
class CVertexBuffer : public IVertexBuffer {
public:

private:
};

/*****************************************************************************************************/
/*                                       NULL頂点バッファクラス定義                                  */
/*****************************************************************************************************/
class CNullVertexBuffer : public IVertexBuffer {
public:
    virtual ~CNullVertexBuffer() {}
};
```

IVertexBufferクラスは、CVertexBufferクラスとCNullVertexBufferクラスのインタフェースとして使用したいので、それらのクラスの基底クラスにし、publicな関数を純粋仮想関数で宣言します。CDXGraphics9クラスが頂点バッファクラスのオブジェクトを生成しようとしたとき、その成否によってCVertexBufferオブジェクトまたはCNullVertexBufferオブジェクトを生成(するように設計)します。アップキャストにより、IVertexBufferへのポインタには、そのどちらも代入することができます。IVertexBufferポインタをとおせば、そのインスタンスがCVertexBufferオブジェクトでもCNullVertexBufferオブジェクトでも、意識することなく操作することができるというわけです。

CVertexBufferクラスがDirect3DVertexBufferオブジェクトをカプセル化するクラスになります。このクラスに、頂点バッファを制御するために必要な機能と属性を追加していきます。
CNullVertexBufferクラスがNull Object役になります。今後、CNullVertexBufferクラスには、IVertexBufferクラスのpublicな関数をすべて「何もしない処理」として実装します。

最終的なクラス相関図は、以下のようになります。



(２)頂点バッファクラスのソースファイル(VertexBuffer.cpp)を以下のように作成しましょう。

－VertexBuffer.cpp－

```
/*
===================================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
===================================================================================================

  【対象ＯＳ】
     Microsoft Windows2000/XP

  【コンパイラ】
```

```
    Microsoft Visual C++ 2005

【プログラム】
    VertexBuffer.cpp
                頂点バッファクラス

【履歴】
    * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

/*************************************************************************************************/
/*                                        インクルードファイル                                   */
/*************************************************************************************************/
#include "VertexBuffer.hpp"
#include <cassert>
```

(３)CVertexBufferクラスに、頂点バッファを管理するのに必要なメンバを追加します。以下のプログラムを適切な場所に追加しましょう。

```
    IDirect3DDevice9*         m_pD3DDevice;
    IDirect3DVertexBuffer9*   m_pVertexBuffer;

    const DWORD               m_FVF;            // FVFフラグ
    const UINT                m_Stride;         // 次の頂点までのバイト数
```

頂点バッファを管理するだけなら、IDirect3DVertexBuffer9だけメンバ変数にすれば良さそうですが、頂点バッファをレンダリングする際に、IDirect3DDevice9、FVFフラグ、次の頂点までのバイト数が必要になるので、これらもメンバ変数に加えておきます。

(４)CVertexBufferクラスのコンストラクタとデストラクタを作成します。以下のプロトタイプをpublicに追加しましょう。

```
    CVertexBuffer(IDirect3DDevice9* pD3DDevice, IDirect3DVertexBuffer9* pVertexBuffer,
                  const DWORD inFVF, const UINT inStride);
    virtual ~CVertexBuffer();
```

コンストラクタでは、メンバ変数を初期化しますが、初期化に必要な情報のうち、外部から受け取らなければならない値を引数としています。コンストラクタの仕様は、以下のとおりです。

CVertexBuffer　コンストラクタ

CVertexBufferオブジェクトを構築します。

試 `CVertexBuffer(IDirect3DDevice9* pD3DDevice, IDirect3DVertexBuffer9* pVertexBuffer, const DWORD inFVF, const UINT inStride);`

pD3DDevice	Direct3DDevice9オブジェクトへのインタフェース
pVertexBuffer	Direct3DVertexBuffer9オブジェクトへのインタフェース
inFVF	FVFフラグ。このフラグで指定された情報を持つ頂点として扱われます
inStride	次の頂点までのバイト数。この間隔で頂点を格納しているものとみなされます

(５)コンストラクタを実装します。以下のプログラムをソースファイルに追加しましょう。

```
/*************************************************************************************************/
/*                                        コンストラクタ                                         */
/*************************************************************************************************/
CVertexBuffer::CVertexBuffer(IDirect3DDevice9* pD3DDevice, IDirect3DVertexBuffer9* pVertexBuffer,
                             const DWORD inFVF, const UINT inStride)
    : m_pD3DDevice(pD3DDevice), m_pVertexBuffer(pVertexBuffer), m_FVF(inFVF), m_Stride(inStride)
{
    assert(m_pD3DDevice != NULL && m_pVertexBuffer != NULL);

    m_pD3DDevice   ->AddRef();
    m_pVertexBuffer->AddRef();
}
```

コンストラクタでは、受け取ったインタフェースのAddRefメソッドを呼び出し、参照カウンタを増やしています。COMインタフェースを別の変数にコピーした場合、その期間が一時的でない場合は、この

ようにした方が安全です。
　この例では、CVertexBufferオブジェクトが存在している間は、各インタフェースが参照しているオブジェクトが解放されなくなります(ただし、不正にReleaseメソッドが呼び出された場合はこの限りではありません)。もちろん、このままでは参照カウンタが増えたままになり、解放されなくなるので、CVertexBufferオブジェクトが破棄されるときに、Releaseメソッドを呼び出し、参照カウンタを減らします。

(６)デストラクタを実装します。以下のプログラムを完成させ、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          デストラクタ                                             */
/*****************************************************************************************************/
CVertexBuffer::~CVertexBuffer()
{
    ※ここは各自考えましょう(m_pVertexBufferを解放します)
    ※ここは各自考えましょう(m_pD3DDeviceを解放します)
}
```

　デストラクタでは、コンストラクタで増やされたインタフェースの参照カウンタを減らし、正しく解放されるようにします。

(７)よく使われる頂点データの構造体を定義します。以下のプログラムをヘッダファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          構造体定義                                               */
/*****************************************************************************************************/
// トランスフォーム済みライティング済みの頂点
struct DXGTLVERTEX {
    float      x, y, z, rhw;    // 頂点座標
    D3DCOLOR   color;           // 頂点色
    float      tu, tv;          // テクスチャ座標

    // コンストラクタ
    DXGTLVERTEX() : x(0.0f), y(0.0f), z(0.0f), rhw(0.0f), color(0), tu(0.0f), tv(0.0f) {}
};

// 未トランスフォームライティング済みの頂点
struct DXGLVERTEX {
    float      x, y, z;         // 頂点座標
    D3DCOLOR   color;           // 頂点色
    float      tu, tv;          // テクスチャ座標

    // コンストラクタ
    DXGLVERTEX() : x(0.0f), y(0.0f), z(0.0f), color(0), tu(0.0f), tv(0.0f) {}
};

// 未トランスフォーム未ライティングの頂点
struct DXGVERTEX {
    float   x, y, z;            // 頂点座標
    float   nx, ny, nz;         // 法線ベクトル
    float   tu, tv;             // テクスチャ座標

    // コンストラクタ
    DXGVERTEX() : x(0.0f), y(0.0f), z(0.0f), nx(0.0f), ny(0.0f), nz(0.0f), tu(0.0f), tv(0.0f) {}
};
```

(８)(７)で定義された構造体のFVFフラグを定義します。以下のプログラムをヘッダファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                            定　数                                                 */
/*****************************************************************************************************/
// FVFフラグ定義
enum { DXGFVF_TLVERTEX = D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1,
       DXGFVF_LVERTEX  = D3DFVF_XYZ    | D3DFVF_DIFFUSE | D3DFVF_TEX1,
       DXGFVF_VERTEX   = D3DFVF_XYZ    | D3DFVF_NORMAL  | D3DFVF_TEX1 };
```

(９)頂点バッファオブジェクトの解放漏れを防ぐため、すべての頂点バッファオブジェクトをCDXGraphics9クラスが管理するようにします。
以下のメンバをCDXGraphics9クラスに追加しましょう。

```
std::list<IVertexBuffer*>   m_VertexBuffer;       // 頂点バッファリスト
```

(10)CDXGraphics9クラスに、頂点バッファオブジェクトを生成するCreateVertexBuffer関数を追加します。以下の仕様を読み、プログラムを完成させて適切な場所に追加しましょう。なお、プロトタイプの追加も忘れずに行いましょう。

CreateVertexBuffer　生成

指定されたサイズおよびフォーマットを格納する頂点バッファオブジェクトを生成し、そのインタフェースを返します。

```
試IVertexBuffer* CreateVertexBuffer(const UINT inSize, const DWORD inFVF,
                                    const UINT inStride);
```

Return	生成された頂点バッファオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります
inSize	バッファのサイズです。最低でも頂点データが１つ格納できる大きさを指定しなければなりませんが、頂点データの倍数である必要はありません
inFVF	FVFフラグ。このフラグで指定された情報を格納するためのバッファが生成されます
inStride	次の頂点までのバイト数。通常は、頂点データのサイズを指定します

```
/*****************************************************************************************************/
/*                                        頂点バッファ生成                                           */
/*****************************************************************************************************/
IVertexBuffer* CDXGraphics9::CreateVertexBuffer(const UINT inSize, const DWORD inFVF, const UINT inStride)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateVertexBuffer)\n");
        return NULL;
    }
#endif

    // 頂点バッファ生成
    IVertexBuffer*          pVertex;
    IDirect3DVertexBuffer9*  pBuffer;
    if(m_pD3DDevice->??????????????????(inSize, 0, inFVF, D3DPOOL_MANAGED, &pBuffer, NULL) == D3D_OK) {
        pVertex = new CVertexBuffer(m_pD3DDevice, pBuffer, inFVF, inStride);
    } else {
        ::OutputDebugString("*** Error - 頂点バッファ生成失敗(CDXGraphics9_CreateVertexBuffer)\n");
        pVertex = new ????????????????();  // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pBuffer);

    m_VertexBuffer.push_back(pVertex);      // 頂点バッファリストに追加する

    return pVertex;
}
```

(11)CDXGraphics9クラスに、頂点バッファオブジェクトを解放するReleaseVertexBuffer関数を追加します。この関数の引数は、解放する頂点バッファのインタフェースです。
以下のプログラムを完成させて適切な場所に追加しましょう。なお、プロトタイプの追加も忘れずに行いましょう。

```
/*****************************************************************************************************/
/*                                        頂点バッファ解放                                           */
/*****************************************************************************************************/
void CDXGraphics9::ReleaseVertexBuffer(IVertexBuffer*& pVertex)
{
    m_VertexBuffer.??????(pVertex);     // 頂点バッファリストから削除する
    delete pVertex;                     // 頂点バッファオブジェクトを解放
    pVertex = NULL;                     // NULLを代入し、無効なポインタであることを明示する
}
```

(12)以下のプログラムは、CDXGraphics9クラスが保持しているすべての頂点バッファオブジェクトを解放するReleaseAllVertexBuffers関数です。足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          全頂点バッファ解放                                         */
/*****************************************************************************************************/
void CDXGraphics9::ReleaseAllVertexBuffers()
{
    for(std::list<IVertexBuffer*>::iterator it = m_VertexBuffer.?????(); it != m_VertexBuffer.???(); it++)
        delete *it;
    m_VertexBuffer.clear();
}
```

(13)CDXGraphics9クラスの解放時に、すべての頂点バッファが解放されるようします。以下のプログラムを適切な場所に追加しましょう。

```
    // 全頂点バッファ解放
    ReleaseAllVertexBuffers();
```