建造物の3次元コンピュータグラフィックス

Production of a computer software showing a three dimensional object

田 縁 正 治

コンピュータグラフィックスにより、3次元の建造物を仮想的に作成する方法を検証した。3次元グラフィックスは2次元グラフィックスとは全く異なった手法が必要であり、コンピュータのハードウエア及びソフトウエアの両面で高い性能が必要である。特にリアルなグラフィックスを作成しようとすると、写真データを利用することが必須であり、この結果コンピュータが負担しなければならない処理量が大幅に増大する。検討した結果、増大した処理量を短い時間で処理するよう効率の良いプログラムを作成するには言語としてVisual C++を使用することが良かった。また、3次元空間を作成し、テクスチャを貼り3次元から2次元に変換するためのプログラムを作成するにはMicrosoft社が無償で提供するDirect Xを使用することが良い方法であることが分かった。ハードウエアに関してはNVIDIA社製のGeFORCE4は十分な性能を持っていた。これらの性能を実際に検証するためのプログラムは宮崎公立大学を建造物として作成した。これは、センター試験を受験するために宮崎公立大学を訪れる高校生に、試験会場内を仮想的に訪れることを支援する目的で作成した。このプログラムを利用した結果、いくつかの改良点が見出されたが、昨年の同様の目的で作成したプログラムに比べてリアルで好評だった。

キーワード：3次元コンピュータグラフィックス、ソフトウエア、仮想空間、大学案内、センター試験

目 次

Ｉ 序　論

近年のパーソナルコンピュータの性能の向上は著しく、以前はワークステーションや大型計算機が行っていた業務をパーソナルコンピュータが処理することが可能となってきた。これに伴い、比較的低価格で導入できるパーソナルコンピュータに対する期待が高まってきた。一方、ソフトウエア開発技法も開発され、ほんの数年前まではBASICやＣ言語が主流であった[1-2]が現在はC++やJAVAやC#など[3-7]が利用されるようになり、オブジェクト指向プログラミングが定着してきた感がある。また、BASICもオブジェクト指向を前面に出した言語製品が登場しプログラマにとっては選択肢が増えソフトウエア開発が楽になったような錯覚に陥る。だが、プログラミングはその言語を利用すること以外にプログラミングの対象の理解が不可欠であることは現在も全く変化していない。また、パーソナルコンピュータは性能が向上したとはいえ、処理量が多い場合はさまざまな工夫を行わないと期待される性能を発揮することができない。

建造物は３次元空間に存在するものだが、これを仮想的にパーソナルコンピュータのディスプレイ上で表現することができればさまざまな応用が考えられる。本稿ではこのようなコンピュータソフトウエアを作成するために必要な手法を調査・テスト・解説をすることを目的とする。その際コンピュータグラフィックスができるだけリアルであることと建造物内をスムーズに移動できるレベルであることを目標とする。この目的を達成しようとするといくつかの問題点がある。表現がリアルになるためには建造物の写真を撮影し、それを利用することが不可欠となる。写真は一般にデータ量が多いので、利用する写真の枚数が増えるとコンピュータの負荷が急激に増大する。更に、建物内をスムーズに移動するには限られた時間内にデータ処理を終了しなければならないことを意味する。また、リアルであるためには実際に３次元空間をコンピュータのメモリ内に作成し、それをディスプレイ上に投影するという処理が必要であるが、これには３次元空間の作成や、３次元から２次元への変換が必要になり、この作成・変換からも処理量が増えることになる。以上述べたような複雑で大量の情報処理を短時間に効率良く行うには通常のソフトウエア作成技法だけでは十分な性能を満たすソフトウエアは作成できない。ここでは通常とは異なる手法を取り入れて建造物を仮想的に作成し、その利用価値を調査することとした。

昨年の本紀要ではコンピュータ内の仮想的な３次元空間に建造物を作成する簡単な方法を調査し、その方法を紹介した。そこでは簡単であることをひとつの重要な目的としていたので、建造物をリアルに表現するという点では不満の残る方法であった。今回は簡単であることを要求しないで、よりリアルであることを目的としてソフトウエア開発技法を開発することとした。したがって、昨年採用したDirect X7 RMモードをDirect X8 IMモードに変更し、開発する言語はVisual BasicからVisual C++と変更した。その結果、パーソナルコンピュータで写真データをふんだんに使用しても１秒間に50フレーム程度の速度で３次元空間を描画することが十分可能であることが判明した。

，このDirect X8は、ランタイムライブラリのみならず、SDK（software developer's kit）すなわちDirect Xを利用してソフトウエアを開発する者のためのライブラリやそのライブラリの説明を書き込んだファイルをMicrosoft社のサイトから無料でダウンロードすることができる。しかし、その説明はすでにDirect Xを理解し利用している者には簡潔で利用しやすいが、これからDirect Xを利用しようとする者にとってはかならずしも容易に理解できるようには記述されていない。ここではDirect Xを容易に理解し利用するための説明も提供することとした。

Ⅱ パーソナルコンピュータの性能

1. グラフィックボード

最近のパーソナルコンピュータの性能を調査し、その限界を知ることが最も良いソフトウエアを作成するには重要な作業である。特にグラフィックスボードの調査は3次元コンピュータグラフィックスにおいては必須である。この調査は3次元グラフィックスで必要とされる処理内容を調査することから始めなければならない[8-9]。この点に関してはすでに詳述したので[10]、ここでは簡単に触れる。3次元グラフィックスで必要とされる処理はメモリ上に仮想的な3次元空間を用意しその中に3角形を組み合わせて立体的な建造物を作成・変形・移動するTransformationや、作成した物体に光を当てるLighting、3次元空間からコンピュータのディスプレイに表示するために2次元に変換するラスタライズ、写真データを貼るテクスチャなどが挙げられる。TransformationとLightingはまとめてジオメトリ演算と呼ばれる。その他はレンダリングと呼ばれる。通常、コンピュータのCPUは汎用的な目的で作成されているので、上述した処理に特化されていない。一方、グラフィックスチップはジオメトリ演算を効率良く行うために作成されていることがある。ハードウエアT&Lを処理するとされるグラフィックチップを搭載したグラフィックカードは一般に3次元空間を作成することに向いていると考えられる。ここではNVIDIA社製のGeFORCE4 TI4200を搭載したAOPEN社製AEOLUS TI4200を使用した。

CPUからグラフィックスボードへのデータ転送は一般にグラフィックボード内と比較してかなり遅いことが多いのでこの点を考慮し、できるだけVideoメモリ内に必要なデータを置いておくような工夫をすることが効率の良いデータ利用方法と言える。このことからVideoメモリの容量が多いグラフィックカードが望ましい。上述した製品は128MBのVideo DDRメモリを持っているので、十分な性能を期待することができる。これからコンピュータを自作する場合は最新の情報を利用してなるべく性能の良いグラフィックボードを選択することが重要である。

2. Direct XとVisual C++

3次元空間をコンピュータのメモリ上に作成し、ディスプレイに表示できるようにその3次元のデータを2次元に変換をする機能は、高度な知識の基に作成されたソフトウエアによってのみ

提供される。幸いこの要求を満たすソフトウエアがマイクロソフト社によって開発され、毎年のように改良されている。このソフトウエアはDirect Xという名前で無償提供されているので、マイクロソフト社のホームページからダウンロードして使用した。ここでは、Direct X8.0を使用した。Direct X8はDirect X7から大きく変化し、3次元を扱うことをより重要視している。つまり、Direct X7では2次元空間を作成するための機能を用意し、その上に3次元空間を作成するための機能を構築していた。しかし、Direct X8では最初から3次元空間を作成する機能を用意し、2次元空間は3次元空間の特別な場合という位置づけとなった。また、新たにさまざまな機能を提供し、ハードウエアがまだ用意していないような機能をソフトウエアが先行して備えてハードウエアの改良を促すような部分も存在する。また、3次元空間内にポリゴンを描く際に頂点バッファ利用することが標準であるように設計が変更されている。これは、頂点バッファを使用しない場合のポリゴンの描画がDrawPrimitiveUPという関数で提供されているのに対して頂点バッファを使用する場合の描画はDrawPrimitiveというより簡単な名前の関数が用意されていることでも分かる。ただし、このDirect X8はあくまでも3次元空間を作成するための基本機能を提供するだけなので、実際に仮想的な3次元空間を作成するようなソフトウエアを必要とする場合はDirect Xを利用するソフトウエアを目的に応じて作成しなければならない。

Direct X8のもうひとつの大きな変更はDirect X7で提供されていたRMモードが廃止され、IMモードに統一されたことである。RMモードは簡単にDirect Xを利用する方法を提供していたが、その機能は限定され処理速度が遅かった。このモードは簡単な方法でDirect X使用する場合には便利で、昨年の本紀要にはVisual BasicとDirect X7 RMモードという組み合わせで3次元空間に建造物を作成する方法を示した。しかし、Direct Xの性能を十分に引き出すにはIMモードを使用した方が良い結果がでるのでRMモードが廃止されたものと思われる。本来Direct XはVisual C++と組み合わせて使用することを前提にして作成されているので、ここではVisual C++とDirect X8 IMモードという組み合わせを採用することとした。Direct Xの利用方法はVisual C++に付属のオンラインヘルプではなく、Direct X8のためのオンラインヘルプを利用しなければならない。この理由は、Visual C++に付属のDirect Xのオンラインヘルプは Dirct X5のためのヘルプだったからである。

Ⅲ　ソフトウエアの作成のための手法

1．座標系

空間を正確に表現するには座標を利用することが良い方法であることはいうまでもない。しかし、座標系は2種類あり、指定を誤ると混乱の元となる。Direct Xではデフォルトで左手系のデカルト座標を用いる。左手系ではディスプレイの左から右へ移動するとx座標が増加する。ディスプレイを下から上へ移動するとy座標が増加する。z座標はディスプレイの手前から向こうに

移動するときに増加する。右手系ではz座標の向きが逆になる。右手系に慣れたプログラマはできることなら右手系で記述したいと望むであろう。Direct X では D3DXMatrixPerspectiveRH 関数や D3DXMatrixOrthoRH 関数や D3DXMatrixLookAtRH 関数が用意されているので左手系を右手系に変換することは不可能ではない。しかし、機能によってはデフォルトで用意されている設定をデフォルトでない設定に変更しなければならないこともあり、決して安心できない。たとえば、後述する三角形の描画においてはカリングモードを変更しなければならない。Direct X では左手系を利用した方が混乱を生じないと思われるので、左手系を薦めることにして、上記の3つの関数の説明はここでは行わない。

2. 色

色の設定はrgb値で行う。つまり、赤の成分と緑の成分と青の成分をそれぞれ指定する。DWORD型の変数に代入することも可能である。

```
DWORD a = 0xff1100;
```

と指定すると赤の成分が0xff = 255、緑の成分が0x11 = 17、青の成分は0と指定したことになる。この方法は16進法で数字を表すときは便利だが、10進法ではあまり都合が良くない。各色の成分を10進法で表現するときはRGB(0, 17, 255)と書くことができる。これは

```
#define RGB(r, g, b) ((DWORD) (((BYTE) (r) | ¥
    ((WORD) (g) << 8)) | ¥
    (((DWORD) (BYTE) (b)) << 16)))
```

と定義されたマクロを利用しており、赤と青の成分の指定位置が上述の方法と入れ替わっていることに注意して使用する必要がある。これはC言語で用意されている方法である。後者の指定方法はCOLORREFという名前で表現されることもある。Direct X では D3DCOLOR__XRGB(r, g, b) や D3DCOLOR__RGBA(r, g, b, a) や D3DCOLOR__ARGB(a, r, g, b) と指定する方法も用意されている。返す値は D3DCOLOR型で、

```
typedef DWORD D3DCOLOR;
```

と定義されているので DWORD型と同じである。r、g、bは赤、緑、青の成分を表す。aはアルファ値を表し、色の透明度を指定するときに利用される。以上の色指定方法では、全て各成分の値の範囲は0から255である。

色の指定において、徐々に明るくなるような色の設定つまりグラデーションを指定したいことがある。このような場合は赤と青と緑の比は同じで明るさだけ異なるようにするので、

```
DWORD c, a = 0xf;
c = a + (a<<8) + (a<<16);
```

といった指定をすると効果的である。徐々に暗くなる影の設定などには是非使いたい手法である。

3．ベクトル

３次元空間の中の最も簡単なデータは点のデータである。点のデータはベクトルとして指定する。Direct Xでは点のデータはD3DXVECTOR3構造体のインスタンスとして表現される。C++ではstructやclassはキーワードであり、構造体はクラスの一種として扱われるのでD3DXVECTOR3はコンストラクタやオーバーロード演算子を持つことができる。実際D3DXVECTOR3はD3DXVECTORのメンバを継承している。D3DXVECTORは

```
typedef struct __D3DVECTOR{float x; float y; float z;} D3DVECTOR;
```

と定義されている。これはCプログラマ用の点データの表現方法を提供する。ここでx、y、zは空間の中の点の座標を表す。C++プログラマはD3DXVECTOR3を利用できるのでプログラミングがより容易になる。D3DXVECTOR3ではコンストラクタ

```
D3DXVECTOR3(FLOAT x, FLOAT y, FLOAT z);
```

が用意されているので、

```
D3DXVECTOR3  a(1.0f, 0.0f, 0.0f);
D3DXVECTOR3  a = D3DXVECTGOR3(1.0f, 0.0f, 0.0f);
```

という記述が可能である。また、＋、－、== などのオーバーロード演算子も用意されているので、ベクトルの演算等に利用できる。ベクトルの成分は個別に

```
a. x = 1.0f; a. y = 2.0f; a. z = 3.0f
```

と記述することも可能である。また、ベクトルの間の加算や比較も可能である。２つのベクトルの内積を計算するためにD3DXVec3Dot関数が用意されている。この関数は

```
FLOAT D3DXVec3Dot(CONST D3DXVECTOR3* pV1,
CONST D3DXVECTOR3* pV2);
```

と定義されているので使用方法は容易に分かる。２つのベクトルの外積のための関数も用意されているが、外積の場合は計算結果がベクトルなので、使用方法が少し異なる。この関数はD3DXVec3Crossという名前で、

```
D3DXVECTOR3* D3DXVec3Cross(D3DXVECTOR3* pOut,
CONST D3DXVECTOR3* pV1, CONST D3DXVECTOR3* pV2);
```

と定義されている。つまり、引数がひとつ増えてその引数の中に計算結果が入ることになる。また、この関数の戻り値は計算結果のポインタになっているので、別の関数の引数とすることができる。これらの関数の他に、D3DXVec3Length、D3DXVec3LengthSq、D3DXVec3Normalizeなど利用価値の高い関数も用意されている。それぞれ、ベクトルの長さ、ベクトルの長さの２乗、ベクトルの正規化を行う関数である。また、補間用にD3DXVec3LerpやD3DXVec3Hermiteも用意されている。

4. 角　度

角度の単位として通常使われているのは度である。これは円の周囲を一周すると360度変化したことにする単位である。360という数字は特に意味のあるものではなく、この単位の作成は根拠を持たない。強いていえば、360という数字は多くの約数を持つということであろう。つまり、2、3、4、5、6、10など多くの数で割り切ることができる。この他にラジアンという単位が使われている。これは円の周囲を一周したら2πラジアン変化したことにする単位で、半径1の円の円周の長さと同じ値を持つ。180度がπラジアンに相当する。90度では円の1/4だから、円周の長さ2πの1/4倍の$\pi/2$が直角に相当する。πはDirect Xでは

```
#define D3DX_PI    ((FLOAT)  3.141592654f)
```

とD3DX_PIという名前になっている。したがって30度をラジアンに変換するコードは

```
30.f * D3DX_PI/180.f
```

となる。この変換は関数が用意されているので、それらを利用しても良い。

```
D3DXToRadian(degree) ((degree) * (D3DX_PI/180.0f))
```

は度からラジアンへ、

```
D3DXToDegree(radian) ((radian) * (180.0f/D3DX_PI))
```

はラジアンから度への変換を行う。

5. 行　列

3次元空間内の点を表現するにはベクトルを利用すれば良い。しかし、点データは単に表現するだけでは有効利用できない。それを変更することができなければならない。点の移動は平行移動と回転に分類することができる。平行移動は単にベクトルの成分に移動量に相当する値を足せば良いが、回転の場合には回転角を基にして回転のための行列を用意して行列とベクトルの積を計算することで処理することができる。簡単のためにxy平面内での回転を例にとるとベクトル(x, y)を角度θだけ回転する場合は

$$(X\ \ Y) = (x, y) * \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

とすることにより変換できる。この考え方を3次元空間に適用することで3次元ベクトルを回転することができる。Direct Xでは平行移動と回転を別の処理としないで一括して処理することとしている。この目的のために行列は3次元行列ではなく4次元行列としている。つまり通常の3次元の回転を扱うための行列の右および下に列と行を追加しこれらを平行移動のために使用することで一括処理を行う。点の変更は上記の移動だけではない。空間の変更も必要となる。3次元空間内の点をコンピュータのディスプレイに表示するために3次元空間から2次元空間への変換する場合もある。

ベクトルの変換をするための行列は、Direct XではD3DXMATRIXのインスタンスとして

宣言する。D3DXMATRIXはベクトルと同様C言語の機能を拡張している。C言語ではD3DMATRIXを

```
typedef struct __D3DMATRIX { union { struct {
    float  _11, _12, _13, _14;
    float  _21, _22, _23, _24;
    float  _31, _32, _33, _34;
    float  _41, _42, _43, _44; } ;
    float m[4][4];     } ; } D3DMATRIX;
```

と定義している。したがって行列の要素は_11などで指定することができる。C++プログラマはD3DXMATRIXを利用できるのでプログラミングがより容易になる。D3DXMATRIXではコンストラクタ

```
D3DXMATRIX(FLOAT _11, FLOAT _12, FLOAT _13, FLOAT _14,
    FLOAT _21, FLOAT _22, FLOAT _23, FLOAT _24,
    FLOAT _31, FLOAT _32, FLOAT _33, FLOAT _34,
    FLOAT _41, FLOAT _42, FLOAT _43, FLOAT _44 );
```

が用意されているので、

```
D3DXMATRIX  a(1.f, 0.f, 0.f, 0.f, 0.f, 1.f, 0.f, 0.f, 0.f, 0.f, 1.f, 0.f, 0.f, 0.f, 0.f, 1.f);
```

という記述が可能である。また、＋、－、== などのオーバーロード演算子も用意されているので、ベクトルの符号を変えたり、行列の要素は個別に値を指定したりすることもできる。行列の成分は16個あり、ひとつの行列を作成するだけで多くのコードを書くことになり扱いにくい。Direct Xではこの点に配慮し、行列を簡単に作成する方法も用意している。

単位行列の場合は

```
D3DXMATRIX mat;
D3DXMatrixIdentity(&mat);
```

とすると引数として与えられた行列が単位行列となるように要素を指定される。具体的には上記の行列aの要素と同じ値を各要素に指定することになる。また、回転行列を簡単に作成するためにD3DXMatrixRotationX、D3DXMatrixRotationY、D3DXMatrixRotationZ、D3DXMatrixRotationAxisなどの関数が用意されている。それぞれx軸の周りの回転、y軸の周りの回転、z軸の周りの回転、任意の軸の周りの回転に対応している。また、特殊な用途ではあるが、D3DXMatrixRotationYawPitchRollという関数も用意されている。これはヨー、ピッチ、ロールを指定するための行列で、舟や飛行機などを描画するときは大いに利用価値があると思われる。これらの回転では、角度は全てラジアン単位で指定する。したがって度単位でデータを用意している場合は、単位の変換を行う必要がある。逆行列はD3DXMatrixInverse関数で設定することができる。D3DXMatrixTransposeは転置行列を生成する。平行移動のための行列

は、D3DXMatrixTranslation関数で作成することができる。

行列は単に物体の移動のためだけに利用されるのではない。視点、つまりコンピュータのディスプレイに表示される像をどの位置から撮影したかを表すことにも利用される。これはカメラの位置と向きとして考えればよい。この指定を行うための関数が用意されている。

D3DXMatrixPerspectiveFovLH(&mat Proj, D3DX__PI/4, fAspect, 1.0f, 100.0f);

第1引数は演算結果、第2引数は視野角をラジアン単位で、第5引数は描画する時の最も近い位置の値、第6引数は描画する時の最も遠い位置の値である。最も近い位置とは、3次元空間内で移動してある建造物に近づいた場合にいつまでその建造物を描画するかを表す。つまり、1と指定すると、距離が1になるまでは描画するが、それより近づくと描画を止める。最も遠い位置はその距離より遠い位置にある建造物は遠くて見えないということにして描画をしない。行列式を計算するためのD3DXMatrixfDeterminant、2つの行列の積を計算するためのD3DXMatrixMultiplyなど計算のための関数も用意されている。

6. テクスチャ

建造物をリアルに作成する良い方法は写真を利用することである。なるべく正面から撮影した写真を建造物の面に貼り付けることでリアルさは格段に上がる。この手法はテクスチャと呼ばれる。テクスチャの大きさは2のべき乗であることが望ましい。つまり、2×2×2×2＝16、2×2×2×2×2＝32などの数字が2のべき乗であり、このような数字が写真データの縦と横の大きさになっていることが望まれる。

テクスチャには座標を設定する必要がある。いま、4角形に1枚の写真をテクスチャとして貼る場合を考える。左上をtu＝0.0、tv＝0.0とし、右下をtu＝1.0、tv＝1.0とすると四角形に貼り付けたときに写真データがぴったり4角形に過不足なく貼られる。この場合、右上は当然tu＝1.0、tv＝0.0とし、左下はtu＝0.0、tv＝1.0としておく必要がある。つまり、左右方向がtu座標であり、上下方向がtv座標である。

テクスチャとする写真データはファイルとして与える方法とリソースとして与える方法がある。どちらにしても

LPDIRECT3DTEXTURE8　　texture;

として、テクスチャ用のポインタ変数を用意しておき、テクスチャを読み込む。ファイルから読み込む場合はD3DXCreateTextureFromFileExA関数を利用する。たとえば

```
D3DXCreateTextureFromFileExA(m__pd3dDevice,"..￥￥Figure￥￥Texture.bmp",
0,0,0,0,D3DFMT__A8R8G8B8,D3DPOOL__MANAGED,D3DX__FILTER__LINEAR,
D3DX__FILTER__LINEAR,0,NULL,NULL,&Texture);
```

と使用する。第2の引数がファイル名を指定している。この例ではファイルが存在フォルダも指定する。つまりパス名で指定することとした。上記の例の場合のフォルダについてもう少し説明

する。今、フォルダaがあるとする。このフォルダの中にFigureという名前のフォルダを作成しその中にテクスチャに使用するファイルTexture.bmpを保存したとする。また、フォルダaの中にbというフォルダを作成し、その中にC++で作成した実行可能ファイルがあるとする。つまり、このフォルダがカレントディレクトリとなる。このフォルダの中からTexture.bmpファイルにアクセスする時は上記のように指定する。もうひとつのテクスチャを利用する方法は、リソースとし写真データを用意して利用する方法である。

```
3DXCreateTextureFromResourceA(m_pd3dDevice,hInst,(LPCTSTR)IDB_BITMAP1,
&texture);
```

とするとリソースの中でIDがIDB_BITMAP1であるデータがテクスチャとして利用可能になる。ここでhInstはこのプログラムのインスタンスを表す。

実際にテクスチャを利用するときはテクスチャを適用する図形を描画する前に

```
m_pd3dDevice->SetTexture(0, texture);
```

と設定する。最後に

```
SAFE_RELEASE(texture);
```

としてテクスチャの使用を終了しなければならない。

7. 頂　点

頂点のデータはその位置ベクトルはもちろんだが、それ以外のデータも含むことができる。たとえば色のデータを設定するとその頂点で指定される多角形がその色で塗りつぶされる。もし、ひとつの三角形の各頂点の色を別の色に指定すると、その三角形の塗りつぶしはグラデーションとなる。すでに説明したテクスチャ座標を指定することもできる。テクスチャ座標についてはすでに説明したので、ここでは頂点のデータの指定方法に限って説明する。

Direct Xでは頂点のデータの与え方はあらかじめ限定されていない。これは柔軟な頂点フォーマットと呼ばれ、プログラマが頂点のデータの与え方を指定できる。この理由は、はじめから頂点のデータのフォーマットを決めておくと、頂点に指定できるデータの全てを指定する方法を用意することになり、不経済だからである。つまり、頂点に指定できる値のすべてを一部の値しか使用しないプログラマまでが指定しなければならないことになる。このような理由のために柔軟な頂点フォーマットが用意されている。

ここでは頂点のデータとして、位置ベクトルと色、そしてテクスチャの座標を指定することにする。これらのデータはまとめてひとつの構造体としておく。つまり、

```
struct BUILDVERTEX
{
    D3DXVECTOR3 p;      //Vertex position
    DWORD       color;  //Vertex color
```

```
    FLOAT          tu, tv ; // Vertex texture coordinates
};
```

とする。ここで、BUILDVERTEXはこの頂点のフォーマットにここでつけた名前である。ビルの頂点のデータをこの方法で指定するつもりでこのように命名した。ここで、D3DXVECTOR3は3次元ベクトルを指定するための構造体である。その他に色のデータとしてDWORD型の変数を1個とテキスチャ座標を指定するためにFLOAT型の変数tuとtvを用意した。この頂点のフォーマットに名前を付けるために

```
#define D3DFVF_BUILDVERTEX
(D3DFVF_XYZ | D3DFVF_DIFFUSE | D3DFVF_TEX1)
```

という1文を入れる。ここで使用したD3DFVF_BUILDVERTEXは上記のような頂点のデータの組つまり上記のような頂点フォーマットに対する名前である。この名前はレンダリングする際に

```
m_pd3dDevice->SetVertexShader(D3DFVF_BUILDVERTEX);
```

と、頂点フォーマットの情報を与えてからレンダリングすることを指示する。

多くの頂点からなるデータを指定する場合、頂点のデータをたびたびグラフィックボードに送るのはあまり効率の良い方法ではない。この問題を解消する方法として、VRAMに頂点のデータをあらかじめ送っておき、描画で頂点のデータが必要になったときに頂点のデータを送るのではなく、頂点のデータがあるVRAM中の位置をデータとして送る方法が用意されている。こうするとCPUからグラフィックボードに送るデータ量を大幅に減らすことができるので効率の良い処理を実現できる。この方法では頂点バッファを利用する。

```
m_pd3dDevice->CreateVertexBuffer(100*sizeof(BUILDVERTEX),
D3DUSAGE_WRITEONLY, D3DFVF_BUILDVERTEX, D3DPOOL_MANAGED,
&m_pBuildVB)))
```

として頂点バッファを作成し、

```
BUILDVERTEX*v;
m_pBuildVB->Lock(0, 0, (BYTE**)&v, 0);
memcpy(&v[0], p, 4*sizeof(BUILDVERTEX));
m_pBuildVB->Unlock();
```

と、Lock関数とUnLock関数の間で頂点バッファにデータを書き込む。実際にはここで多くの頂点のデータを次々に書き込むことになる。上記の例では、頂点のデータをv[0]だけに書き込んでいるが、v[1]、v[2]などに書き込むことにより連続したデータとして扱うことができ、多くのデータを一括して扱うことを可能にする。最後は

```
SAFE_RELEASE(m_pBuildVB)
```

で使用を終了する必要がある。

8. ポリゴン

物体を作成して描画するには物体のデータをコンピュータに与える必要がある。Direct Xでは、3角形を基本としている。すなわち、3角形を組み合わせてさまざまな3次元物体を作成する。この組み合わされた多角形はポリゴンと呼ばれる。したがって、物体のデータを与えるときは物体を3角形に分解し、その3角形の頂点の情報を与えることになる。この頂点のデータを与えるときの順番にも意味がある。頂点を順番にたどったときに時計回りに見える方をその三角形の表とし、反対側を裏としている。デフォルトではDirect Xは表のみを描画する。

3角形の裏を描画するには、カリングモードをセットする。これはSetRenderState関数を利用することで設定できる。

m__pd3dDevice->SetRenderState(D3DRS__CULLMODE, D3DCULL__NONE);

とするとカリングしないで、表裏の両面を描画する。

m__pd3dDevice->SetRenderState(D3DRS__CULLMODE, D3DCULL__CW);

とすると裏面のみを描画し、

m__pd3dDevice->SetRenderState(D3DRS__CULLMODE, D3DCULL__CCW);

とすると表の面のみを描画する。

多くの建物はおおむね4角柱だから、側面は4角形である。4角形は2つの三角形を組み合わせることとする。したがって、側面だけで8個の三角形を描画することになる。円柱や球の場合も小さい3角形を組み合わせて近似的に表現する。

物体を作成して描画するためにDirect X8は2つの方法を用意している。ひとつは、ポリゴンを描画したいときに頂点座標を与えてポリゴンを描画する方法である。これはDrawPrimitiveUpという関数を呼ぶことで利用される。もうひとつは、あらかじめ頂点データをVRAMに転送しておき必要になったらこのデータの保存場所とポリゴン数指定してポリゴンを描画する方法である。つまり、頂点バッファを利用する方法である。これはDrawPrimitive関数を呼ぶことで利用される。前者の利用例を示す。

m__pd3dDevice->DrawPrimitiveUP(D3DPT__TRIANGLEFAN, 2, m__face.v, sizeof(FACEVERTEX));

第1の引数は頂点のデータから3角形を描画する際のデータの利用方法、第2の引数は描画する3角形の個数、第3引数は頂点データの位置、最後の引数はひとつの頂点のデータの大きさをバイト単位で与える。次に後者の例を示す。

m__pd3dDevice->DrawPrimitive(D3DPT__TRIANGLESTRIP, 0, 4);

これは頂点バッファを利用する場合の方法で簡単な書式で済む。第2の引数はあらかじめ与えた頂点のデータの中で、利用する初めのデータ位置を表す。今は0だから、最初のデータから順に利用することを意味する。第3の引数は描画する三角形の数を表す。

9. 円 柱

宮崎公立大学では24本の円柱が中庭を囲むように立てられており、これが大学の概観を決定する重要な要素となっている。したがって、この柱をリアルに描画することは重要である。3次元空間の物体は基本的にはすべて3角形を組み合わせたポリゴンとして描画する。したがって、円柱は正確には表現できない。そこで多角形で近似することとした。いくつかの条件で試した結果、円柱は底面が20個の頂点を持つ多角形で近似することで十分リアルだった。20個の頂点を使用すると、円柱の側面は上下の底面にそれぞれ20個の頂点を作成し、これらを利用して40個の三角形で近似することになる。ひとつの頂点はDirect Xが提供するベクトル型であるD3DXVECTOR3型のデータを一組と、色を指定するためのDWORD型のデータひとつ、テクスチャ座標を表すFLOAT型のデータ2つを必要とする。これらのデータを1本の円柱で40組持つので、もし24本の円柱すべてが別々にデータを持つとすると、かなりのデータ量となる。これらのデータを1フレームごとに主記憶装置からグラフィックカードに転送すると、データ転送がボトルネックになる恐れがある。パーソナルコンピュータの構造としては主記憶装置とグラフィックカードの間のバスはあまり転送速度が速くないようになっているので、上記の方法はあまり良い方法とはいえない。そこで、この頂点のデータはグラフィックカードの中にあるVRAMに保存することで、主記憶装置からのデータ転送を抑えることで全体の処理速度を上げることが期待できる。このような理由から円柱の頂点のデータを保存するために頂点バッファを利用することとした。

もうひとつの問題はテクスチャである。表面に写真をテクスチャとして貼るのだが、このテクスチャのデータを1フレームごとに主記憶からVRAMに転送していたのでは、上記の頂点バッファの効果がなくなる。そこで、テクスチャのデータもVRAMに置くことになる。このテクスチャのデータは写真であるためデータ量が多い。そこでなるべく品質が良くてデータ量が少なくなるような工夫が必要となる。通常写真は晴れた日に撮影すると光の量が多いので品質の良い写真が取れるように思える。そこで円柱を晴れた日に撮影した。しかし実際に使用してみると、本ソフトウエアはおおむね直射日光が当たらない廊下や室内を描画することが多いので、不自然になった。24本の円柱で同じ写真データを利用することを考えたので、直射日光の当たる柱と当たらない柱のデータを用意することはデータをいたずらに増加することになると判断し、曇りの日に円柱をデジタルカメラで撮影し利用した。データ量を減らす他の工夫も行った。円柱には横に4本の黒い線が入っているので、1本の円柱は5個の部分から成り立っていると考えた。そこで、円柱の写真はこの5個の部分のひとつ分を用意して上下にずらしながら利用して1本の円柱を描いても十分リアルであることになる。さらにデータ量を減らすために円柱のひとつの部分の写真も上下ほぼ同じであるので、上下反転して使用すればデータ量は半分になる。同じことを左右で行うとさらにデータ量は半分になる。つまり、1本の円柱の写真データは全体の1/20のデータで済むことになった。

次に実際に円柱を作成するときに使用したコードを示す。まず

```
#define NUM__COLUMVERTEX 20
```

として円柱の底面の点の数を決めた。そして、頂点バッファに側面のデータを代入した。

```
m__pColumnVB->Lock(0, 0,(BYTE**) &v, 0);
for(DWORD i=0;i<NUM__COLUMVERTEX; i++){
    FLOAT theta  =(2*D3DX__PI*i)/(NUM__COLUMVERTEX-1);
    v[2*i+0].p  = D3DXVECTOR3(0.25f*sinf(theta), 2.6f, 0.25f*cosf(theta));
    v[2*i+0].tu =((FLOAT)i)*2.0f/(NUM__COLUMVERTEX-1);
    v[2*i+0].tv = 0.0f;
    v[2*i+1].p  = D3DXVECTOR3(0.25f*sinf(theta), 0.0f, 0.25f*cosf(theta));
    v[2*i+1].tu =((FLOAT)i)*2.0f/(NUM__COLUMVERTEX-1);
    v[2*i+1].tv = 10.0f;
}
m__pColumnVB->Unlock();
```

このように底面を20分割してテクスチャを使用した結果かなりリアルな円柱となった。

Ⅳ　ソフトウエア使用時の評価

作成したソフトウエアは数人の人の試用後の批判を基に改良して宮崎公立大学で行われたセンター試験の前日に使用した。このソフトウエアは、試験前日は大学の建物の中に入れないので試験室をコンピュータ上で案内するという目的で作成した。数人の高校生に評価を求めたところ、とてもリアルであるという感想を得た。ビデオ撮影した映像と誤解した高校生もいた。これは予想したことであり、作成した3Dがビデオカメラを片手に撮影した映像と誤解されてはコンピュータの可能性を高校生に伝えることができないので、わざと庭はコンピュータによる簡単な3Dの様にしておいた。ビデオと誤解された場合はこの点を指摘して誤解を解いた。

試験室では、窓が描かれているので、窓の外の景色を描画する必要があった。この点は、今回は省略して室内から窓を撮影した時に写った窓の外の景色をそのまま利用した。したがって、窓を見る位置を変えた時にリアルさに欠けることとなったが、この点を指摘する高校生はいなかった。高校生の注意は自分の受験する机の方にあり、窓の外の景色にあまり注意が払われなかったからのようである。すべてをリアルに表現することが最も良い3D映像ではないことが分かった。今後この教訓を生かして3D映像を作成することとしたい。

3次元空間内の移動に関する評価では、良い点と悪い点があった。建物の中で移動する時に、階段の昇り降りは自動としたことは好評だった。階段を昇ったり、踊り場での方向転換をしたりすると無用な負担をかけるのでこの方法を採用した。通常の移動では、キーボードを使用するよ

うにしたが、これはあまり好評ではなかった。マウスを使用する方が良いとの評価だった。次回はマウスで移動するよう変更することを検討中である。

受験番号を入力して試験室の中の受験する机までたどりつけると、歓声が上がることもあり、好評だった。また、試験室の外から受験する机まで誘導するための矢印を表示しておいたが、これも好評だった。

テクスチャは、なるべく数を制限してコンピュータの負担を抑えることとした。壁は、ひとつの写真を何度も使った。これは使用者にもそれと分かるようだが、特に不快感はなさそうだった。講義室の入り口は前後に2つある。これも同じ写真を利用した。前の入り口と後ろの入り口を鏡像関係になるように使用したが、これは気づかれなかった。同じ写真を左右反転して使用していることを説明して初めて気づいた人もいた。2つの講義室で同じ写真を利用したが、これも気づかれなかった。また、エレベータの写真も1階と2階で同じ写真を使用したが、これも気づかれなかった。1階では入り口の周りがレンガ調であるのに、2階では白壁であることを説明して初めて気づいた。以上のことから、テクスチャは工夫次第でかなり枚数を減らすことが可能である。最初に見せる像がきれいに作りこまれていれば他も同程度に作りこんであるという先入観が働くようだ。

大学正面の建物の写真はきれいに撮影することが困難だった。近くから撮影しようとすると、下から見上げた写真となるので、歪んだ写真となり、テクスチャに利用するときに不便だった。一方、遠くから撮影しようとすると、木や時計台など、邪魔物が存在した。今回はある程度距離を置いて建物の正面の写真を撮影した。その結果、樹木が写りこんだが、左右ほとんど対称な建物であることを利用して写真データに手を加えて樹木を取り除くよう努めた。完全に取り除くことができなかったが、使用した結果は、だれも不完全な部分を指摘することはなかった。テクスチャは利用するとかなりの効果を発揮するので、たとえ不完全な写真でも良いから大いに利用した方が良い。

その他のさまざまな指摘を列挙しておく。講義室の黒板に文字を書いておくとよりリアルだとの指摘があった。また、建造物の中を歩く靴音を入れると良いという指摘や、机の上のシールに受験番号を書くことを指摘した人もいた。今回は壁にはテクスチャを貼らなかったので、壁にテクスチャを貼ることを次回は考える予定である。また、机の天板などに今回は厚みを持たせなかったが、この点を指摘した人もいた。目的地に近づくとガイドの矢印の色が変わるとより分かりやすいという指摘もあった。ドアから中に入るところが分かりにくいといった指摘や、人を手前に描き、その人を操作するようにすると操作する時の感覚がつかみやすいという指摘もあった。これらの指摘は今後の改良に役立つよう心がける予定である。

結論として、ソフトウエアはVisual C++とDirect X8の組合せを利用して作成し、ハードウエアはGeFORCE4を利用すると良い結果が得られた。通常のソフトウエアはCPUが情報処理を行い、その結果をディスプレイに表示するのだが、ここではCPUは全体の流れの制御を

行い、多くの処理はグラフィックチップが分担して行う。また、グラフィックスチップが必要なデータはVRAMに置くことでCPUからグラフィックカードへのデータ転送がボトルネックにならないような工夫を行った。この結果、頂点の数の多い円柱を数本描き、リアルになるようにテクスチャを多用しても、150から200fps(frame per second)と、かなりの速度で建造物の3次元グラフィックスをパーソナルコンピュータで描くことができた。したがって、多くの応用が期待できる。

謝　辞

本研究において、宮崎学術振興財団より一部資金援助を頂いた。

参考文献

1. 林晴比古、1997年、「Visual Basic 5.0入門教室」、翔泳社
2. 柴田望洋、1998年、「明快C言語入門編」、SOFTBANK
3. 林晴比古、1999年、「新Visual C++6.0入門」、SOFTBANK
4. 瀬戸　遥、2000年、「Visual C++6.0入門教室」、翔泳社
5. 河西朝雄、2001年、「標準Javaプログラミングブック」、技術評論社
6. 柏原正三、2002年、「はじめてのJava完全入門」、技術評論社
7. 林晴比古、2002年、「新Java言語入門、シニア編」、SOFTBANK
8. PC Japan、2001年、第6巻、第7号、190-195頁、SOFTBANK PC Japan
9. PC Japan、2002年、第7巻、第11号、134-155頁、SOFTBANK PC Japan
10. 田縁正治、2001年、「3DCGによる大学案内ソフトウエアの作成」、宮崎公立大学人文学部紀要、第9巻、第1号、43-58頁