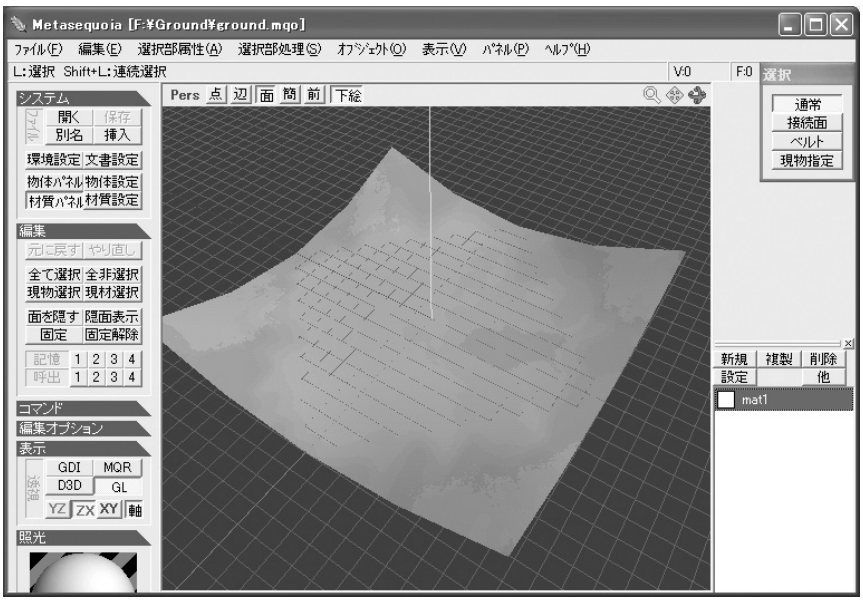
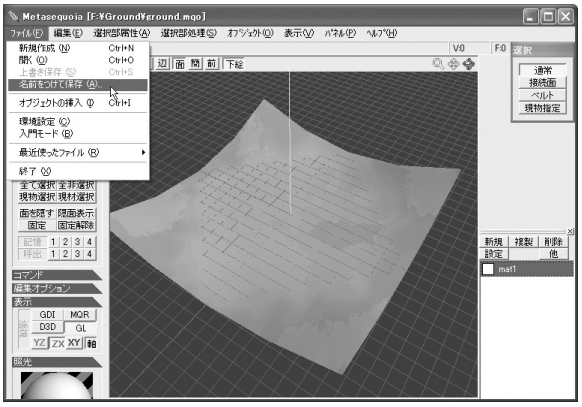
３Ｄモデリングデータをプログラムへ実装する。

地形データを編集する。



DirectX 用の .x 形式で保存する。



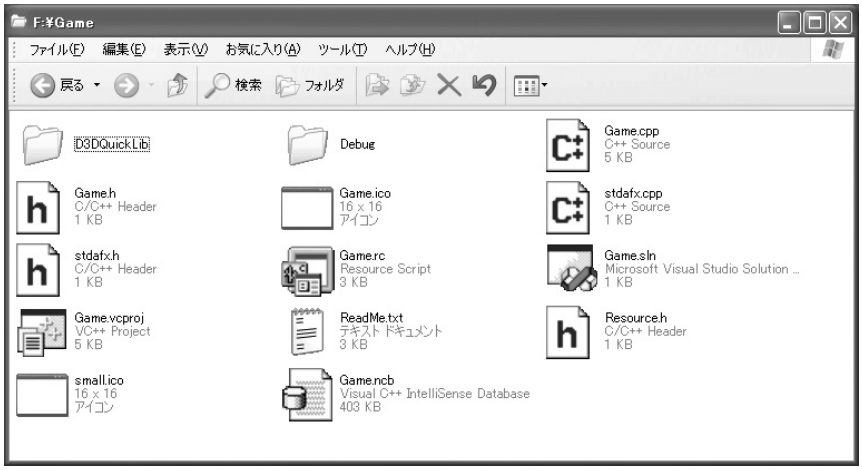
名前を付けて保存。

保存形式に、(.x) を選択して保存する。

ここでは、ground.x というファイル名で保存していますので、以下の解説では、このファイル名を使用します。

データを保存すると、このようになります。
（使用するテクスチャも同じフォルダに保存する必要があります。）

D3DQuickLib を実装したアプリケーションのプロジェクトフォルダを開きます。



このフォルダにある、
「Debug」フォルダを開きます。

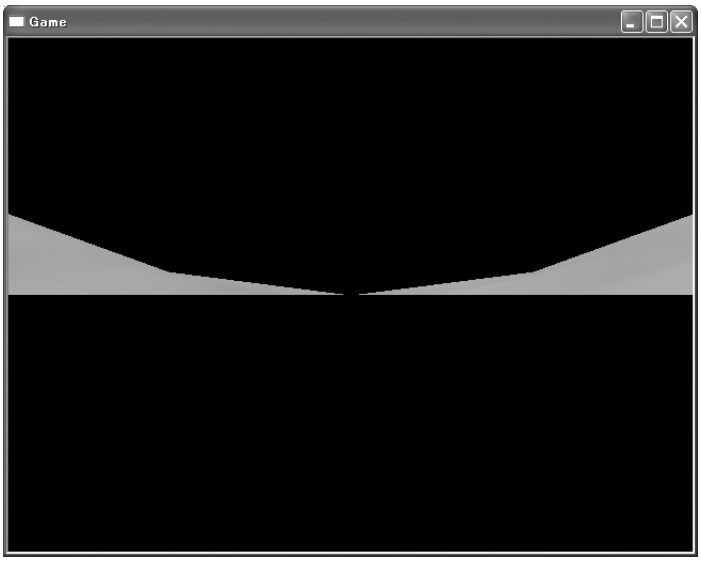
すると、ウィンドウはこのようになっているはずです。

ここに先ほど保存した
メッシュ(.x)形式のデータを
フォルダごと貼り付けます。

ここでは、ground というフォルダの中に左図のようなファイルが入っているものとします。

ファイルは、
ground.x　メッシュ形式データ
ground_skin.bmp　テクスチャファイル。

プログラムリスト１を参照に、プログラムを改造します。



すると､先ほどの地形が表示されるようになりました。

ただ、カメラ位置が地表にくっついている為、モデルが見えにくい状態にあります。

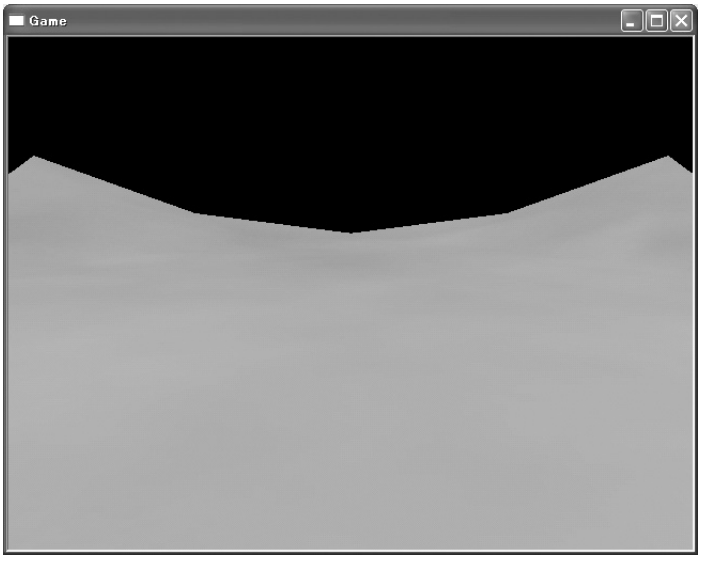
次に、カメラ位置を設定します。

プログラムリスト２を参考に、プログラムを改造します。

ポイントは、以下の部分。CD3DEnv が内部で使用しているビュー行列を取得して、取得されたビュー行列に対して、カメラの場所と向きを DirectX で決められた方式で設定します。

```
D3DXMATRIX     *pView;
pView = g_pD3DEnv->GetSystemView();
D3DXMatrixLookAtLH(pView,&D3DXVECTOR3(0,2.0f,-10.0f),
                         &D3DXVECTOR3( 0.0f, 0.0f, 0.0f ),
                         &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ) );
```

D3DXMatrixLookAtLH 関数については、DirectX 付属のドキュメントを参照して下さい。
（日本語訳もダウンロード可能です。）



これは、以下の設定でモデルを表示させたものです。

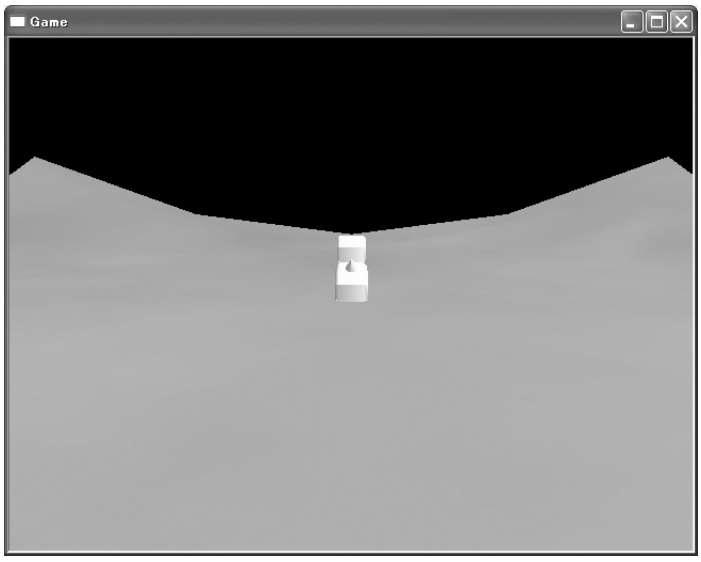
カメラ位置　x:0.0　y:2.0　z:-10.0
カメラ視点　x:0.0　y:0.0　z:0.0
カメラ上方向　x:0.0　y:1.0　z:0.0

カメラ上方向は文字通りカメラの上になる方向をベクトルで示したものです。カメラのひねりを表現するのに使用します。

地形にキャラクタを表示する。

表示させるキャラクタのモデルを用意します。
ここでは、先ほどの地形と同じフォルダに、
「dog.x」というファイルを用意しています。

プログラムリスト３を参考に、プログラムを改造します。改造後実行すると、次のようになります。



地形の中央付近に、「dog.x」が表示されました。
表示されているだけで何らの動きもありません。

次に、このキャラクターを動かします。
プログラムリスト４を参考に、プログラムを改造します。ポイントは以下の３箇所。

ポイント①　キャラクタ位置を示すベクトル型グローバル変数の宣言。

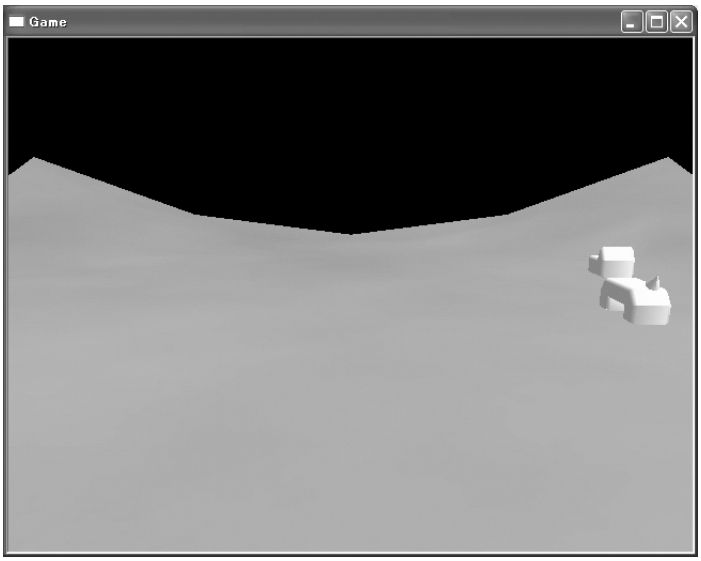
```
D3DXVECTOR3 g_vPos = D3DXVECTOR3(0,0,0);
```

ポイント②　キー入力に応じて位置データを更新

```
if (g_pD3DEnv->GetDI8KeyState(DIK_LEFT)){
      g_vPos.x -= 0.1f * timeElapsed;
}
if (g_pD3DEnv->GetDI8KeyState(DIK_RIGHT)){
      g_vPos.x += 0.1f * timeElapsed;
}
if (g_pD3DEnv->GetDI8KeyState(DIK_UP)){
      g_vPos.z += 0.1f * timeElapsed;
}
if (g_pD3DEnv->GetDI8KeyState(DIK_DOWN)){
      g_vPos.z -= 0.1f * timeElapsed;
}
```

ポイント③　キャラクタ位置の情報からワールド行列を算出

```
D3DXMatrixTranslation(&matWorld,g_vPos.x,g_vPos.y,g_vPos.z);
lpd3ddev->SetTransform( D3DTS_WORLD, &matWorld);
```



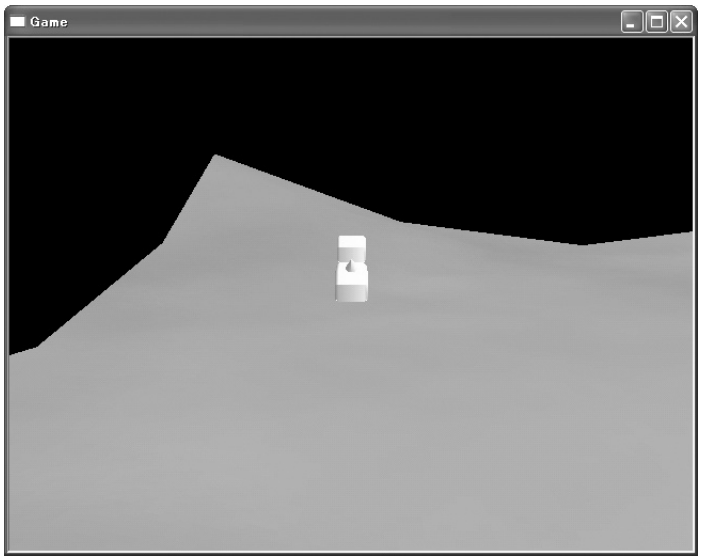
キャラクタが動くようになりました。
ただ、カメラが動かないので、市販のゲームのように､動きに応じて背景がスクロールされるような事がありません。

少し見にくいですね。

カメラを動かします。プログラムリスト５を参考に、プログラムリスト２の改造で手を加えたカメラ位置の設定を少し改造します。

ポイントは、以下の部分。

```
D3DXMatrixLookAtLH(pView,&D3DXVECTOR3(g_vPos.x,g_vPos.y+2.0f,g_vPos.z-10.0f),
                         &D3DXVECTOR3( g_vPos.x, g_vPos.y, g_vPos.z ),
                         &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ) );
```

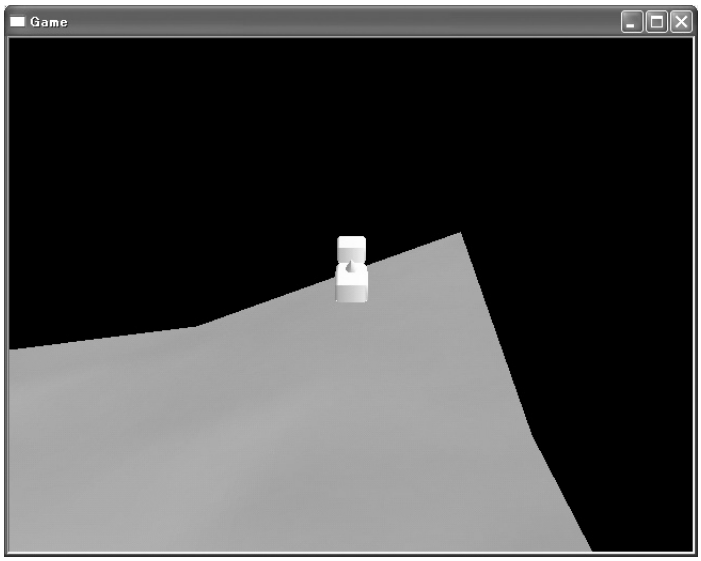


これでキャラクタは常に画面の中央です。

ただ、この地形。起伏がある為に、キャラクタが地形にめりこんでしまいます。

次に、プログラムリスト６を参考に、プログラムを改造し、高度判定を入れます。
ポイントは以下の部分。

```
D3DXVECTOR3 vecMin, vecMax, vecNormal;
FLOAT fAlt,fDist;
g_pFloor->GetBoundingBox(&vecMin, &vecMax);
g_pFloor->ProbeTheGroundAltitude(&g_vPos,&vecMin,&vecMax,&vecNormal,&fAlt,&fDist);
g_vPos.y = fAlt;
```

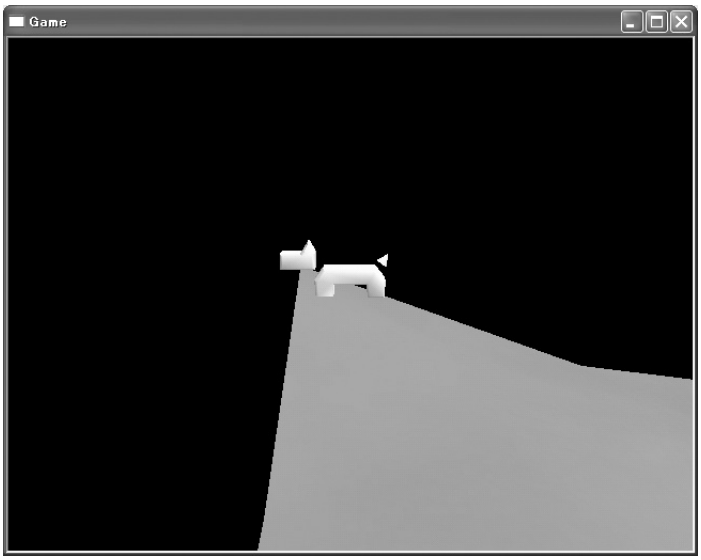


ProbeTheGroundAltitude は、D3DQuickLib が提供する、CFloor クラスのメンバで、メッシュの指定された座標における高度を判定し、返すメソッド。ポリゴンのあたり判定は処理が重いので、処理対象とするポリゴンを、指定した立方体内に限定する機能がある。しかし、今回、大してポリゴンを使用していないので、常に地形全体を示す立方体を用意している。ここでは地形モデルのオブジェクトから、バウンディングボックス（全体を囲む最小の立方体）を取得し、それを当たり判定を行う範囲として使用している。

最後にキャラの向き、プログラムリスト７を参考に、プログラムを改造。
ポイントは以下の部分。

```
D3DXMatrixTranslation(&matWorld,g_vPos.x,g_vPos.y,g_vPos.z);
D3DXMATRIX matRotation;
D3DXMatrixRotationY(&matRotation,g_fAngle);
matWorld = matRotation * matWorld;
lpd3ddev->SetTransform( D3DTS_WORLD, &matWorld);
```

キャラクターを表示する前に、まず移動の為の行列を作り、次に回転の為の行列を算出。続いて、その二つの行列を合成して、（回転→移動）を行う行列を作り、ワールド行列として設定している。
回転行列を作る為には、Direct3D が提供する関数の、D3DXMatrixRotationY を使用している。詳細は、DirectX のドキュメントを参照されたい。



回転の計算に使用している、g_fAngle は角度を表すグローバル変数で、キー入力を行った時に、数値を更新している。

詳細はプログラムリストを参照されたい。