平成１４年度　卒業論文

３Ｄテクスチャを用いた
画像表示の質感の向上

奈良女子大学理学部情報科学科

数理情報学講座　和田研究室

伊佐　友江

はじめに

本研究は，DeltaViewer で３D テクスチャを使用することにより，画像表示における質感の向上を可能にした。今現在では、テクスチャマッピング機能は DeltaViewer に組み込まれ、実用化の段階にまで至っている．ここで述べる DeltaViewer とは、共焦点レーザー顕微鏡等から得られる連続断面画像から計算によって物体の表面を再構築し、表示するためのアプリケーションプログラムのことである．DeltaViewer の画像表示の質感を向上させる為に、本研究では、３D テクスチャマッピングのプログラムの作成、DeltaViewer で３D テクスチャを使用しない画像と使用した画像との比較、物体の再構築に必要な時間との関係の考察を行った。

３D テクスチャの使用を考えた理由としては，テクスチャマッピングの「もとの画像を直接貼付けることができる」という機能を利用することで，DeltaViewer の画像の粗さを改善させられるのではないかと考えたからである．従来の DeltaViewer は，OpenGL グラフィックライブラリを用いて小さな三角形をいくつも描き，それらを組み合わせることで３D 画像を構築していたが，それぞれの三角形の色を頂点のカラー間で補間して描画していた為，細部がぼやけて見えた．pixel 値を小さくしてしまえばそれぞれの三角形も小さくなるため，細かい描画は可能かもしれないが，処理に時間がかかってしまった．そこで，テクスチャマッピングを行うことを考えたわけだが、テクスチャマッピングと一言でいっても，非常に複雑な処理が必要であり，使用に際してもプログラム作成上の選択や制限がある．実際に実験を行うまでには，テクスチャマッピングの原理，手順，使用するにあたっての制限を調べる必要があった．

本論文では，１章で研究目的を述べ，２，３，４章でテクスチャマッピングを行う前に必要となる知識をまとめている．５ 章，６章では，DeltaViewer で３D テクスチャを使用した場合，使用しない場合，処理時間等を比較し，それぞれの関係を考察する．

目次

１．研究目的

DeltaViewer とはアプリケーションプログラムで，共焦点レーザー顕微鏡等から得られる連続断面画像から計算によって物体の表面を再構築し，それをコンピュータ画面に表示する，というものである．

物体表面の再構築は，OpenGL グラフィックライブラリを用いて小さな三角形をいくつも描き，それらを組み合わせるという方法をとっている．それぞれの三角形は境界で近似され，より滑らかに見える様になっており，glMaterial()内で diffuse（拡散光）0.7，emission（放射光）0.3 の比率で昭光処理を行うことで三角形の頂点の色を指定し，頂点のカラー間で補間を行うことで３Ｄ画像に色をつけている．

しかし，この方法では以下のような問題が生じる．

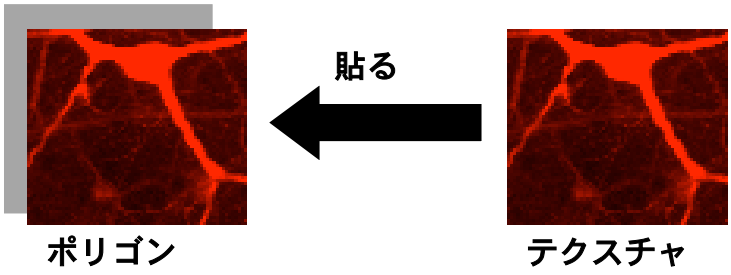
１．それぞれの三角形は頂点のカラー間で補間して描画している為，三角形の中に模様はなく，細かい描写ができず，細部がぼやけて見えてしまう．

２．pixel 値を小さくしてしまえばそれぞれの三角形も小さくなるため，細かい描画は可能かもしれないが，処理に時間がかかってしまう．「処理時間」と「画像の細かさ」の間で，ある程度妥協する必要がある．

そこで，処理時間を短くするために pixel 値を大きくした場合でも， DeltaViewer の画像がぼやけてしまうのを防ぐためには， ３Ｄ テクスチマッピングを使用すればよいのではないかと考えた．

テクスチャマッピングとは， ３Ｄ の物体に対してテクスチャ画像をシールのように貼付けることができ，物体をリアルに表現することを可能にする．

（下図：テクスチャマッピングのイメージ）



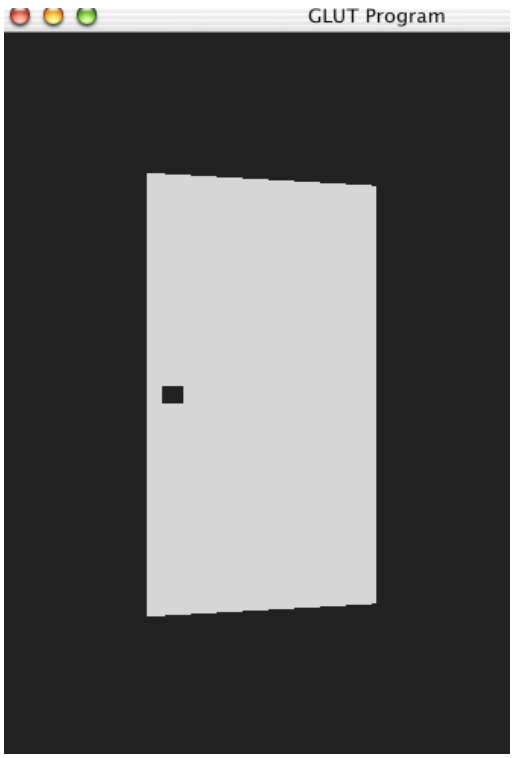
本研究では，処理時間を短くするために pixel 値を大きくした場合でも DeltaViewer の画像がぼやけてしまうのを防ぎ，よりリアルな物体を表現することを目的として，テクスチャマッピングの原理，手順，使用するにあたっての制限を調べ，実験を行った．

２　テクスチャマッピングの原理

２.１　２Ｄテクスチャマッピング

テクスチャマッピングというと、一般的には２Ｄ テクスチャマッピングのことを指す。２ 次元の平面上にある絵柄や模様などを ３ 次元曲面や平面の上に貼り付けることをマッピングといい、貼り付ける模様のことをテクスチャという。２Ｄ テクスチャは回転、平行移動、拡大縮小、射影などの各種の変換を施すことにより、歪みを生じさせることができる。2D テクスチャを ３ 次元曲面などに写像することを２Ｄ テクスチャマッピング(texture mappmg)といい、物体をよりリアルに表現することができ、建築物や植物等、多くのモデルを自然に仕上げるにはこの手法が必要になる。

下の画像は，OpenGL を使って描いた簡単なドアの絵（左下）と、２Ｄ テクスチャマッピングを行い、デジタルカメラで撮ったドアの画像を直接貼付けたもの（右下）である． ２枚の画像を比較した所，２Ｄ テクスチャマッピングした方（右下）が，よりリアルなものになっていることを確認できる。



OpenGL を使って描いたドア

テクスチャマッピングしたドア

２.２　３Ｄテクスチャマッピング

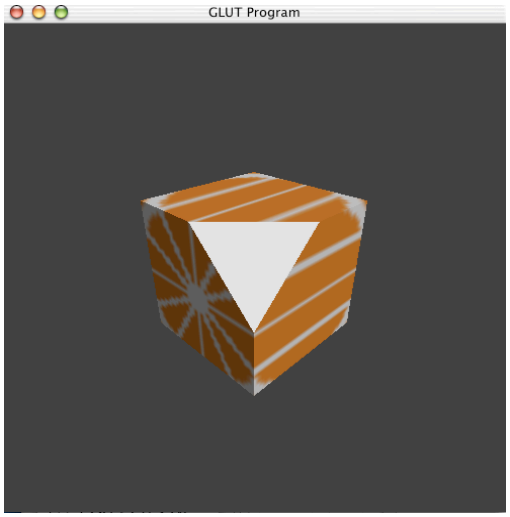
３Ｄ テクスチャマッピングとは、３Ｄ の物体に対し、三次元の画像（３Ｄ テクスチャ）を貼付けることで，３Ｄ テクスチャとは，長方形の２Ｄ テクスチャイメージが層をなしたものと考えることができる．ただし，メモリ内ではその長方形の２Ｄテクスチャは連続して配置され，幅,高さ,奥行きが２の累乗の直方体でなければならない．３Ｄ テクスチャの利点としては、フラクタルや３次元ノイズなどを利用した雲や炎、木目や大理石模様などの切った断面までもちゃんとマッピングできることである。そのため、３Ｄ テクスチャは医学や地球科学の分野のレンダリングに最も良く使われており，医学分野では CT スキャンや MRI のなどの層をあらわす際に，地学の研究員は地層を模型化する際に使用されている。

◇３Ｄテクスチャがどのようなものか，実際に見てみる．

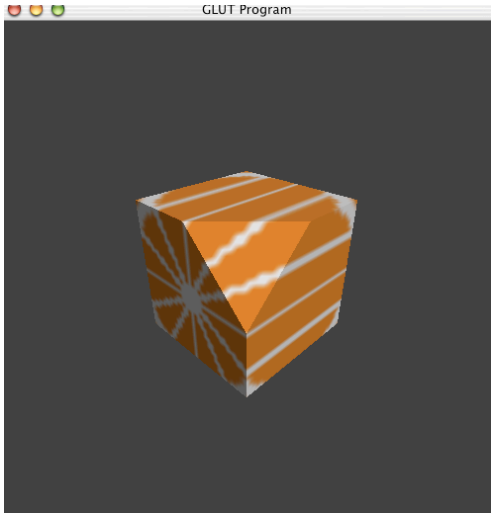
みかんを立方体に切断した絵を用意する。左下は２Ｄ テクスチャのみで作成されたもの．右下は３Ｄテクスチャを使用したものである．

左下の画像を斜めに切断すると、（２Ｄ テクスチャは中身が存在しないため）切り口が白くなってしまう。それに対し、３Ｄ テクスチャを使用すると、切り口までちゃんとマッピングすることができる。このように，中身の情報を扱いたい時に３Ｄテクスチャが有効となる．

２Ｄテクスチャ



３Ｄテクスチャ



（注）３Ｄテクスチャはどこの場所をを切断しても，中身は存在している．

DeltaViewer では，共焦点レーザー顕微鏡等から得られる連続断面画像を３D テクスチャとして，再構築された３D 画像の表面に貼付けたいと考えている．以下で，OpenGL を使った３D テクスチャマッピングについて詳しく述べたいと思う．

３　テクスチャマッピングの手順

３.1 テクスチャマッピングの流れ

ここでは，テクスチャマッピングの実行に必要な手順を簡単に紹介する．

1. 必要なテクスチャの個数 n を決める。
2. glTexGen()によって、現在テクスチャオブジェクトに使用されていない n 個分のテクスチャを指定するための番号を，システムから割り当ててもらう。
3. glBindTexture()によって、テクスチャオブジェクトの作成と使用を行う．
4. 何らかの方法で（ファイルから読み込むか、計算によって生成するなど）メインメモリ内の配列にテクスチャデータを用意する。（OpenGL には TIFF，JPG などの読み込み関数がないのでアプリケーション側で用意する）
5. glTexImage{1,2,3}D によって、このテクスチャのデータをアプリケーションからグラフィックシステムのテクスチャメモリへ送り込む。ただし，画像は２の累乗でなければならない．
6. glTexParameter*()および glTexEnv*()を用いて、このテクスチャの使用条件を指定する。
7. 複数のテクスチャがある場合はそれぞれについて３から６のステップを繰り返す。

ここまでが準備段階となり、テクスチャマッピングされたポリゴンを描くためには

a. glBindTexture()によって、テクスチャの番号を指定する。
b. glBegin()~glEnd()によってポリゴンを描画する際、glTexCoord2f()を各 glVertex*()の直前に行い、ポリゴンの各頂点に対するテクスチャの座標を指定する。

という手順を踏むことになる。なお、ライティングに必要な情報も合わせて考えると、glVertex*()で指定される各頂点に対し、最大で

- n テクスチャ座標(glTexCoord*())
- n 材質指定(glMaterialf*())

n 法線の指定(glNormal*())
の 3 つのデータが付随することになる。

３.2 テクスチャマッピング手順の詳細

前にテクスチャマッピングがどのような流れで行われるのかを簡単に紹介した．テクスチャマッピングとは，つまり，テクスチャの作成と登録，テクスチャ画像を配列に読み込み、その画像に貼付け方法を指定するためのパラメータの設定といった幾つかの手続きをすることである。そこで，以下で順を追って詳しく説明する．

３.2.1 テクスチャの作成

テクスチャマッピングを使用するには，まずテクスチャの作成を行う必要がある．テクスチャを構成する各画素はテクセルと呼ばれ、byte 型 1 次元配列内に左下を起点としてテクセルのカラー値を格納する。貼り付ける画像（２Ｄテクスチャ）は、image[x][y][0..2]　のような配列に記憶する。テクセルフォーマットとして，配列の第 3 添え字が 0,1,2 の順に、RGB 輝度値，もしくは RGBA 輝度値を使用することができる．各輝度は 0～255 を範囲とする符号無し 8 ビット整数値で表され、(R,G,B)=(255,255,255) が白色を、(R,G,B)=(0,0,0) が黒色を意味する。テクセルフォーマットとして RGBA 輝度値を選択した場合は、各テクスチャにアルファ値を持たせることができる。 アルファ値は不透明度を表す係数で、アルファ値 1.0 の時にテクセルは非透過になり、アルファ値 0 の時にテクセルは透明となる。 テクスチャのアルファ値をポリゴン描画時に反映するために Enable メソッドによって GL_BLEND を有効化する必要があるので注意する。

３,2,2 テクスチャオブジェクトの作成

テクスチャを作成した後は、テクスチャオブジェクトの作成を行なう。テクスチャマッピングを行うアプリケーションはレンダリング中多くのテクスチャを切り替えており，複数のテクスチャの切り替えと，テクスチャ管理を容易にするためにテクスチャオブジェクトを使用する必要がある．テクスチャオブジェクトはテクスチャイメージを保持するデータ領域で、3D グラフィクスコンテキスト内に保持されている。同一名で新しいテクスチャを登録したり、ユーザが削除を指示しない限り保持される為、初期化処理時に 1 度だけ作成を行なうと良い。ユーザはテクスチャオブジェクトを切り替えることで、使用テクスチャを変更できる。

glGenTexure でテクスチャオブジェクト名を割り当て，テクスチャオブジェクト

の作成・切り替えは、BindTexture メソッドの引数にテクスチャ名を指定して行う。テクスチャ名とはユーザが任意に指定可能な int 型の識別子である。BindTexture メソッドがコールされると指定されたテクスチャ名を持つテクスチャオブジェクトが有効となる。もし、テクスチャ名に該当するテクスチャオブジェクトが存在しないときは、新規テクスチャオブジェクトが作成される。

3.2.3 テクスチャの登録

テクスチャオブジェクトの作成が完了したら，glTexImage2D でテクスチャの登録を行う．glTexImage2D パラメタは画像のサイズ，型式，位置，その他の特性を指定する．tiff 等の画像ファイルを利用するには画像配列に展開しておく必要がある。次の関数で、マップする画像を指定する。

```
glTexImage2D ( GL_TEXTURE_2D,level, layNum, Width,
        Height, border, GL_RGB, GL_UNSIGNED_BYTE, Image);
```

GL_TEXTURE_2D は 2 次元のマッピングをすること、level はミップマップ処理をしない場合０とする。layNum は画像のレイヤの数で RGB 画像の場合 3。width と height は画像の幅と高さで、画素数単位で指定する。norder は境界線の幅で通常 0。GL_RGB は画像の形式、GL_UNSIGNED_BYTE は符号なし画像データの形式である。最後がテクスチャ画像の配列へのポインタ指定。

また、glPixelStorei でメモリへの格納形式を指定する。

```
glPixelStorei(GL_UNPACK_ALIGNMENT,1);
```

３Ｄテクスチャの場合は？

３Ｄテクスチャは，image[x][y][z][0..2]のような配列に記憶され，

```
glTexImage３D ( GL_TEXTURE_ ３D,level, layNum, Width,
Height, depth, border, GL_RGB, GL_UNSIGNED_BYTE, Image);
```

となり，上記の網かけ部分が変化しただけ．使用方法としては
２Ｄテクスチャとほぼ同じと考えてよい．

ここで，ミップマップ処理という言葉が出てきたが，どのようなものかを次で簡単に説明する．

◇ミップマップ処理

マッピングする対象の図形の大きさが変化する場合、同じ画像を拡大縮小して表示すると不自然になる。そこで、OpneGL では、複数の画像(図３.2.3)を用意して

おき、対象のサイズに一番近い画像を利用することができる。この手法をミップマップ処理という。glTexImage2D の第二パラメータ 0 は一番大きな画像（レベル 0）を意味し，一番大きな画像サイズが 128*128 の場合、レベル 1 で 64*64、レベル 2 として 32*32 の画像を glTexImage2D で指定することができる。

これらのパラメタを指定する時は，テクスチャマップの解像度に 1 回，パラメタ level, width, height, image の値を変更して glTexImage2D()を呼び出す．さらにミップマップ処理したテクスチャを有効にするにはフィルタ処理の方法を選択することが必要である．フィルタ処理については 2.5.5 の高度なテクスチャの設定のテクスチャフィルタで説明する．

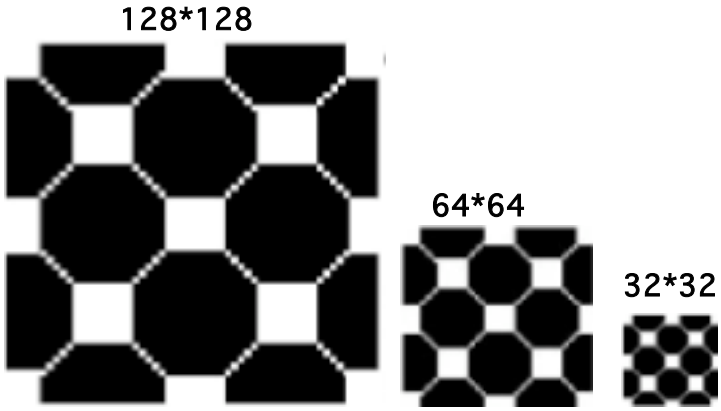


図 3.2.3 ミップマップ処理で使用する複数の画像

3.2.4　テクスチャ付きポリゴンの描画

テクスチャオブジェクトにテクスチャを登録した後は，テクスチャマッピングを有効化してテクスチャ付きポリゴンを描画する．ポリゴンへの貼付け位置を制御するのはテクスチャ座標値で，ポリゴンを構成する頂点毎に glTexCoord2f(s , t)で指定する．テクスチャ座標は，テクスチャに対して横方向を s 座標、縦方向を t 座標とする座標系で、テクスチャ左下位置が（ s, t ）=（ 0.0, 0.0 ）、テクスチャ右上位置が（ s, t ）=（ 1.0, 1.0 ）となる。貼付けは、glTexCoord{1,2,3,4}で現在のテクスチャ座標をセットし，glVertex を呼び出して頂点座標を指定する。

ただし，ポリゴン描画を行う前に，glEnable()でテクスチャマッピング機能を有効化し，BindTexture を用いて使用するテクスチャを指定する必要がある．

3.2.5 高度なテクスチャ設定

ポリゴン描画を行う前に，ポリゴンへの貼り付け方法を指定することも可能であ

る。指定する値としては、主にラッピングモード・テクスチャフィルタ・テクスチャ関数の 3 つがある.

◇ラッピングモード

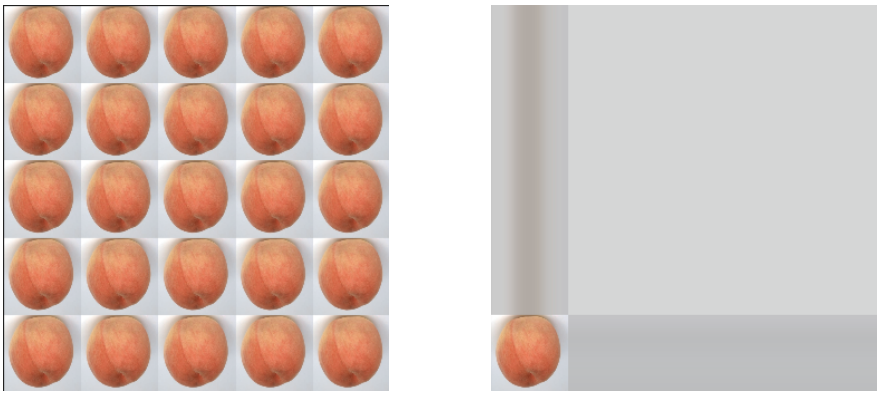
テクスチャ座標値を 0.0～1.0 の範囲外に指定した場合の貼り付け方法を指定でき，glTexParameterf で各種のパラメータを設定する。GL_TEXTURE_WRAP_S(T)は、s,（t）方向に対して繰り返し貼付けを行うか（GL_REPAET)、打ち切る(GL_CLAMP)かを指定する。

使用例）

```
glTexParameteri ( GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT  );
glTexParameteri ( GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_  CLAMP );
```

＜テクスチャ座標値を 0.0～5.0 に指定して貼付けた画像＞

GL_REPAET　　GL_CLAMP



◇テクスチャフィルタ

ここでは，ポリゴンへの貼り付け時に行なうテクスチャの拡大・縮小に使用するフィルタ処理の指定方法を説明する。テクスチャがポリゴンや面にマップされてスクリーン座標に変換されると，テクスチャの個々のテクセルは最終的な画像の個々のピクセルをは対応しなくなる.

そこで，どのように平均化し，または補間するのかを指定する必要がある.

・GL_NEAREST・・・ピクセルの中心に最も近い座標のテクセルが拡大，縮小に使用される.

・GL_LINEAR・・・ピクセルの中心に最も近い２×２配列の加重線形平均が拡大縮小に使用される.

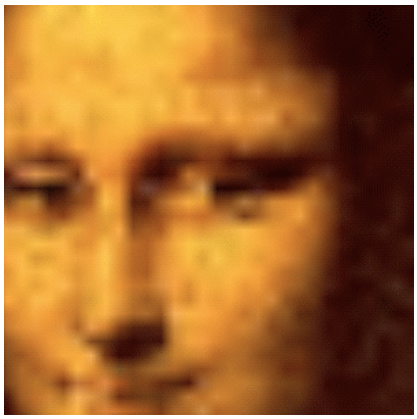
GL_LINEAR よりも GL_NEAREST の方が計算に時間を要さないため処理は高速だが，滑らかな外観を持つのは GL_LINEAR の方である.

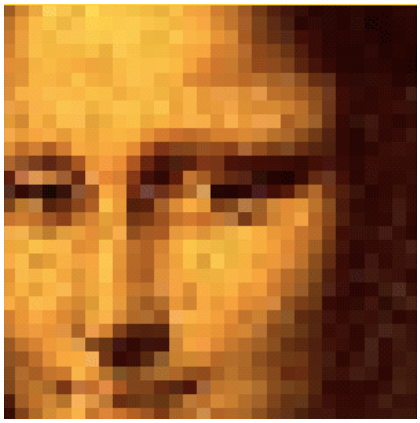
使用例）

glTexParameter(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR　);

glTexParameter(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST　);

GL_LINEAR　　GL_NEAREST





◇テクスチャ関数

　テクセルカラーとポリゴンカラーから描画色を生成する際に,glTexEnv()に適切な引数を供給して，４つのテクスチャ関数の中から，使用するテクスチャ関数を指定する。

・ GL_REPLACE・・・テクスチャカラーをそのまま描画色として表示する.

・ GL_MODULATE・・ポリゴンカラーとテクセルカラーを乗算もしくは加減する.

・ GL_BLEND・・・テクスチャ貼り付け前のポリゴンカラー値とテクセルカラーを混合する.

・ GL_DECAL・・・テクセルカラーをそのまま使用するが、アルファ値に関してはポリゴンで指定されたアルファ値を使用する。
アルファがない場合はテクセルカラーで置き換える.

使用例)

glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE，GL_REPLACE　);

注意事項：

ラッピングモード・テクスチャフィルタ・テクスチャ関数の値はテクスチャオブジェクト内に保持されないため、BindTexture メソッドによるテクスチャオブジェクトの切替では値が変更されない。テクスチャ毎に設定を変更する場合に、再設定が必要となる。

４．3D テクスチャを使用するにあたっての制限

４.1　３Ｄテクスチャを使用するにあたって

Macintosh で３Ｄテクスチャを扱うためにはいくつかの制限がある。
そこで、制限を説明するにあたって知っておかなければならないキーワード、『OpenGL のバージョン』、『グラフィックカードの種類』について説明する。

4.1.1　OpenGL のバージョン

OpenGL は、上位互換のライブラリである。以前のバージョンに含まれる標準機能は、以降のバージョンで使用可能である。例えば、OpenGL 1.1 の頂点配列機能は、OpenGL 1.2 以降でも使用できる。
＜＜OpenGL バージョンの詳細は付録に掲載＞＞
※OpenGL1.2 で『拡張機能の GL_EXT_texture3D が標準機能となった』とあるので，OpenGL1.2 以上で３Ｄテクスチャが扱えることになる.

4.1.2　グラフィックカードの種類

グラフィックカードとは、絵や文字を画面（ディスプレイ）に表示するための処理を行うパーツの事である。画面にどれだけ綺麗な絵を表現出来るか、どれだけ高度なグラフィックを高速に動かす事が出来るかなどは、このパーツの性能による。３D グラフィックなどの高度なものではなく、普通の絵や文字を画面に表示する範囲であれば、それほど大した機能は必要ないが，パソコンでゲームをやったり、DVD ビデオを見たりするのであれば、やはり高性能なグラフィックカードは必須となって来る.

グラフィックカードのメーカーには、その中心となる「グラフィックチップ」だけを配給しているメーカーと、カード全体を作っているメーカーの２つがある。グラフィックチップだけを配給しているメーカーの場合、そのグラフィックチップを使ったグラフィックカードなら、違う会社が作ったものでも同じような性能になる。

GeForce 系 グラフィックチップ 搭載カード	NVIDIA 社の開発したグラフィックチップで，特に３D グラフィックの表示性能に優れている．NVIDIA 社はチップのみをメーカーに配給しているので、グラフィックカードは各社から様々な種類のものが発売されている。 安いものからハイスペックなものまで各種揃っており多くの種類がある。現在最も普及している。

RADEON シリーズ	ATI 社の発売したグラフィックチップ．GeForce シリーズに対抗する性能を持つ 。3D グラフィックに強く、また，DVD などの映像表示に定評があり、それを生かして テレビ/ビデオの再生・録画・編集などの機能を１つにまとめたカードも発売されていて人気となっている。 値段は少し高め。
MILLENIUM G シリーズ	Matrox 社の発売したグラフィックカード．「G400」「G450」「G550」などがある。２D（平面）の画像処理とシャープでクッキリした画質に定評があり、根強い人気がある。しかし，３Ｄグラフィックの表示に弱いのが欠点となっている。ビジネス用のパソコンをメインターゲットとしている。
KYRO 系 グラフィックチップ 搭載カード	ST Microelectronics 社の発売したグラフィックチップ．GeForce に対抗する能力があると言われており、しかも，同レベルの GeForce より安い。しかし，あまり普及しないまま、市場から姿を消した.
VOODOO シリーズ	3dfx 社の開発した、かつて３Ｄグラフィックにおいて定番だったグラフィックカード。しかし，GeForce 登場後は不振で、ついに会社も GeForce の nVIDIA 社に買収され、開発・サポートも停止した。以前は人気のグラフィックカードだった為、今でも使用されているパソコンは多い。
RIVA128(TNT) シリーズ	nVIDIA 社が GeForce の前に開発していたグラフィックチップ。 今でもオンボードや廉価版の VGA に使用されている。
サベージ 2000 (S2000)	S3 社が開発したグラフィックチップ．ドライバの不備などにより GeForce に敗退した。しかし低価格のため、オンボードのグラフィック機能によく使われている。

4.2　３Ｄテクスチャを扱うための条件

Macintosh で３Ｄテクスチャを扱うための条件を以下で述べる。

4.2.1　OpenGL のバージョン

３Ｄテクスチャを扱うためには、

・OpenGL バージョン 1.2 以上（３Ｄテクスチャが標準装備されている）
・OpenGL バージョン 1.２以下であったとしても、拡張機能として３Ｄ
　テクスチャ機能（GL_EXT_texture3D）が実装されている
ことが条件となる。

しかし、グラフィックチップに OpenGL バージョン 1.2 以上が装備され、３Ｄテクスチャが扱える環境になっていたとしても、グラフィックチップをグラフィックカードに組み込む際に３Ｄ テクスチャ機能を取り外してしまう場合もあるため、OpenGL のバージョンだけで３Ｄ テクスチャが扱えるかどうか判断するのは非常に困難である。

4.2.2　３Ｄテクスチャが扱えるグラフィックカード

実際にどのグラフィックカードなら３Ｄ テクスチャが扱えるのか実験し、調査した所、以下の種類に限られることがわかった.

３Ｄテクスチャ使用可能なグラフィックカード

nVIDIA 社：GeForce　３，GeForce　4Ti，GeForce　4GO

ATI 社　：RADEON9700，　RADEON　9000，
RADEON　7500，RADEON，　Mobility　RADEON　9000，
Mobility　RADEON　7500，Mobility　RADEON

4.2.3　VRAM の容量

VRAM とは、グラフィックカードについているメモリのことで，パソコン本体のメモリ（メインメモリ）と同じく一時的にデータを保存する場所である.

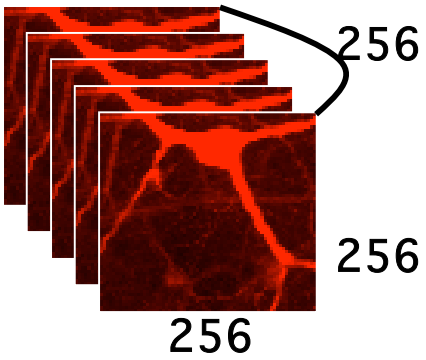
画面に３Ｄ グラフィックなどの高度な画像を表示する場合、その際に必要となるデータ量は大きくなる．グラフィックチップで処理を終えたデータはこのビデオメモリに保存され随時使われて行くが、データ量の多いグラフィックを表示しようとして、しかしメモリが少なくてデータを保存しきれなかった場合、足りなかった分が表現できなくなってしまう。こうなると、特定の所だけ表示されないとか、模様

が一部なくなるとか、色が抜け落ちるとかいう状況が起こり、動作も不安定になってしまう．ビデオメモリ（VRAM）が多いほど、細かい３D 画像や DVD 映像などの高度なグラフィックでも、スムーズに不良もなく表示する事が出来る。

◇OpenGL でテクスチャを使用するには・・・

OpenGL でテクスチャを使用するには，テクスチャサイズが非常に重要になってくる．というのも，３D テクスチャ画像は、消費するメモリ量が非常に大きい為，３D テクスチャを扱える環境が整っていたとしても，VRAM の容量が少ないと３D テクスチャを扱うことができないからである。

EX）256×256×256 サイズの３D テクスチャを用意し、どれくらいのメモリ量を消費するのか計算する。（1pixel =32bit = ４バイト ）



256×256×256×4
= 64×1024×1024
= 64（MB）

一辺が 256 の３D テクスチャは、
64MB のメモリ量を消費することが判明

このサイズの３D テクスチャを扱うためには、64MB より大きい VRAM を用意しなければならない。つまり、グラフィックカードの VRAM を確認した上で、３D テクスチャ画像のサイズを決めることが必要になる。

（注意）DeltaViewer では、幅、高さ、奥行きいずれかの大きさが２５６を超えた場合、テクスチャサイズを 256 以下に縮小し、扱うように設定している。

そこで，希望する内部フォーマットのテクスチャを OpenGL が使用できるかをプログラムがより正確に確認できる機能として，テクスチャプロキシがある．

◇テクスチャプロキシ

テクスチャ画像のプロキシ（代用）を使用することで，リソースが充分であるかを確認する．glTexImage3D()を呼び出した後，glGetTexLevelParameter()でテクスチャの状態変数を確認する．テクスチャプロキシに割り当てるリソースが不十分な場合は，幅，高さ，境界の幅，要素の解像度を表すテクスチャ状態変数はゼロにセットされる．

使用例）

```
GLint format;
glTexImage3D(GL_PROXY_TEXTURE_3D, 0, GL_RGBA8, 64, 64, 64,
```

```
                                    0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);
  glGetTexLevelParameteriv(GL_PROXY _ TEXTURE_3D, 0,
                                        GL_TEXTURE_INTERNAL_FORMAT, &format)  ;
```

4．3　3Ｄテクスチャ対応グラフィックカード表

次の表は、Macintosh の製品にどのグラフィックカードが装備され、３Ｄ テクスチャが扱えるかどうか、VRAM の容量等を詳しくまとめた表である。

	グラフィックカード	３Ｄテクスチャ使用可	VRAM 容量
1	RADEON 9700 Pro	○	
2	RADEON 9000 Pro	○	64MB
3	RADEON 7500	○	32MB
4	RADEON	○	32MB
5	RAGE 128 Ultra	×	16MB
6	RAGE 128 Pro	×	16MB
7	RAGE 128	×	16MB
8	RAGE Pro Turbo	×	6MB
9	Mobility RADEON 9000	○	64MB or 32MB
10	Mobility RADEON7500	○	32MB or 16MB
11	Mobility RADEON	○	16MB
12	RAGE Mobility 128	×	8MB
13	RAGE Mobility	×	4MB
14	RAGE LT Pro	×	4MB
15	GeForce 4GO	○	64MB or 32MB
16	GeForce 4MX	×	64MB or 32MB
17	GeForce 4Ti	○	128MB
18	GeForce 3	○	64MB
19	GeForce 2MX	×	32MB or 16MB

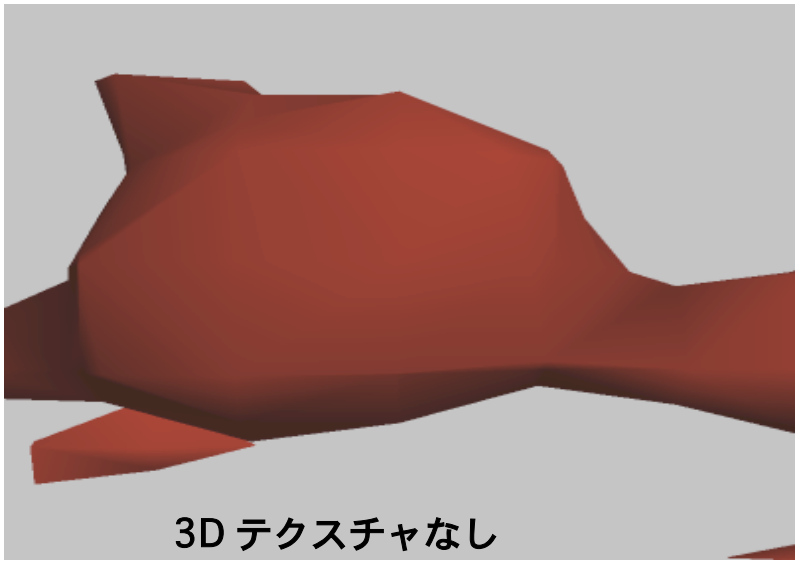
5．３Ｄテクスチャマッピングを用いた実験

3D テクスチャマッピングのプログラムを作成し、DeitaVewer を使って、共焦点レーザー顕微鏡から得られる連続断面画像画像の物体の表面を再構築する．３Ｄテクスチャは使わず、従来の DeltaViewer でそれぞれの三角形の頂点のカラー間を補間して描いた画像と，３Ｄテクスチャを使用して描いた画像を比較する．

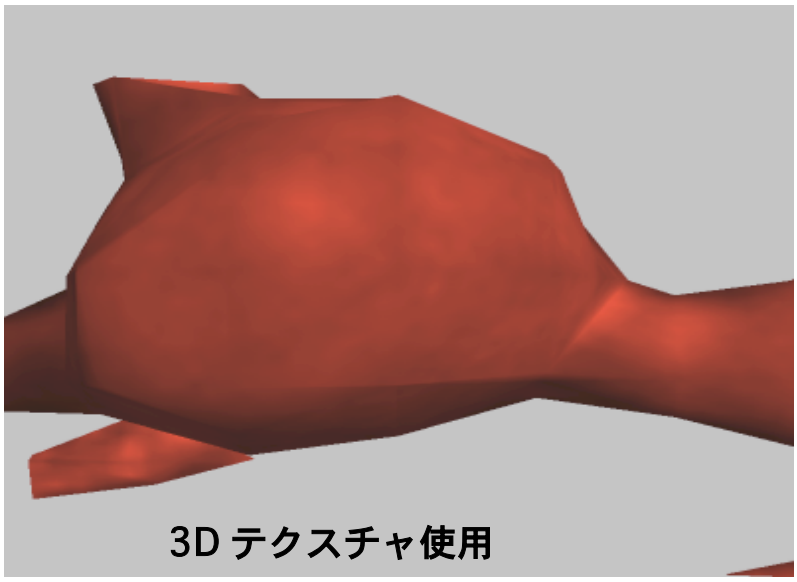
画像は，１０pixel，５pixel，２pixel，１pixel のものを使用する．それぞれの場合で，テクスチャを使用しない場合と使用した場合とを比較し，再構築し終わるまでにどれだけの時間がかかるのかも計測した．

◇10pixel

再構築にかかる時間　＝　１秒



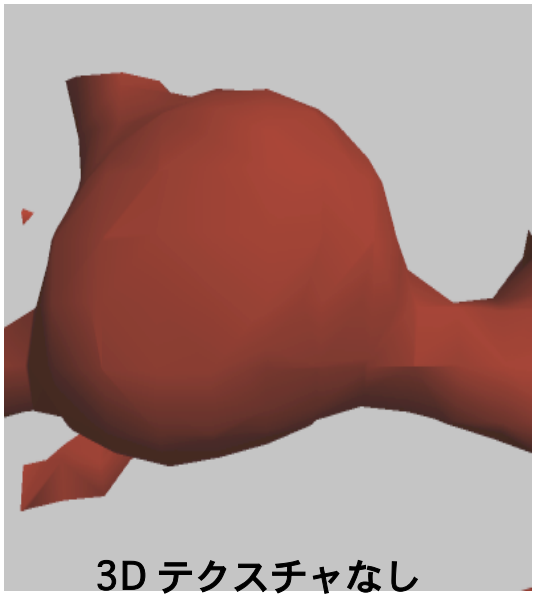
3D テクスチャなし



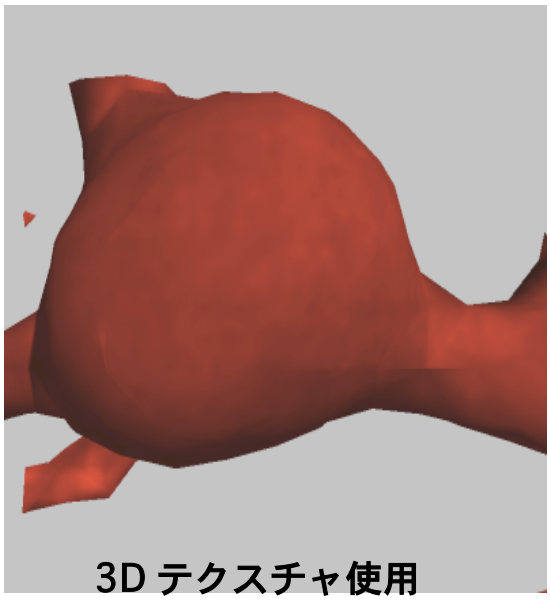
3D テクスチャ使用

◇ 5 pixel

再構築にかかる時間　＝　２秒



3D テクスチャなし

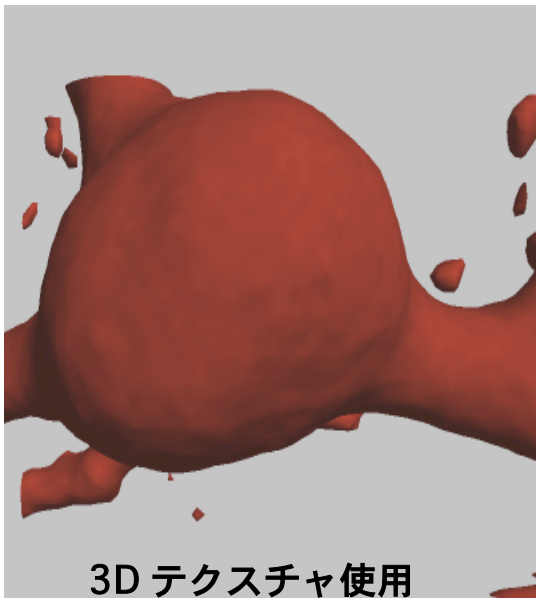


3D テクスチャ使用

◇ 2 pixel

再構築にかかる時間　＝　６秒

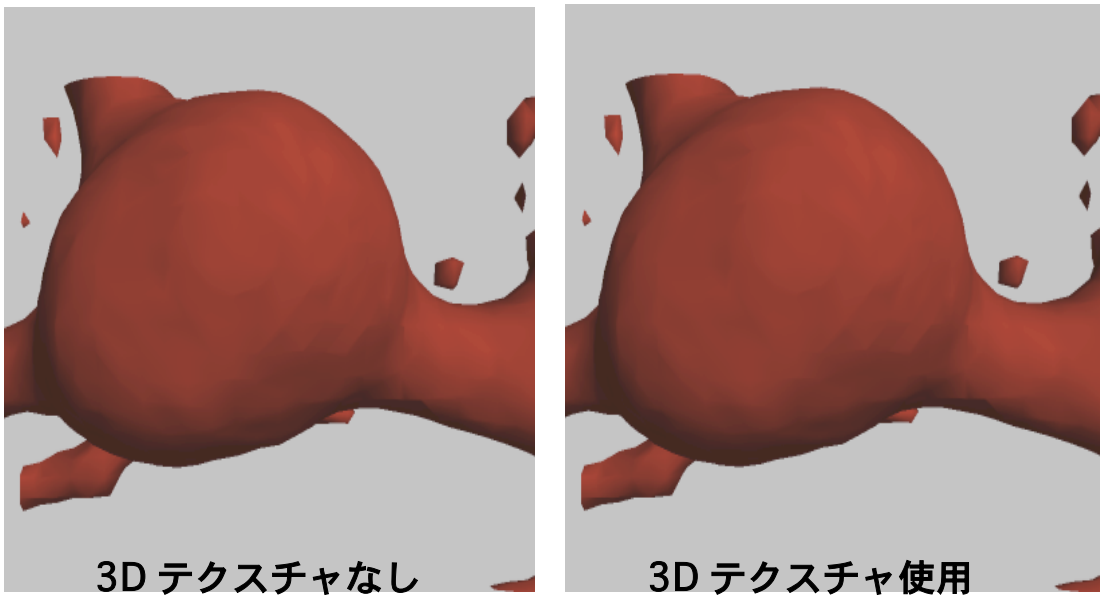
3D テクスチャなし



3D テクスチャ使用

◇１pixel

再構築にかかる時間　＝　４６秒



3D テクスチャなし　　3D テクスチャ使用

6．結果

3D テクスチャを使用しない場合と使用した場合の画像，再構築にかかる時間を比較した．

10pixel の場合

３Ｄ テクスチャなしの時よりも，３Ｄ テクスチャを使用した方がより細かい描画ができていた．再構築にかかる時間は１秒と，処理時間はほとんどかからなかった．結果、３Ｄ テクスチャを用いることで、明らかに画像表示における質感の向上が見られた。

5 pixel の場合

時間も２秒と，ほとんどかからない．画像としても，３Ｄ テクスチャを使用した方が細かい描画ができている．画像表示における質感の向上は見られた．

2 pixel の場合

３Ｄ テクスチャを使用しなくても，使用しても，画像表示においてほとんど変化はない．

1 pixel の場合

３Ｄテクスチャを使用しなくても，使用しても，全く画像表示において変化はない．ただ，再構築までに４６秒かかり，リアルな画像と言えるが，時間がかかり過ぎる．

これらの比較の結果，10pixel，5pixel の時は３Ｄテクスチャを使用することで，画像表示における質感の向上が見られた．２pixel,1pixel の時は，３Ｄ テクスチャを使用してもしなくても，画像表示における質感の向上は見られない．しかし，時間がかかり過ぎてしまう．よって，Pixel 値が大きいときほど，３Ｄ テクスチャを使用することで大きな効果が得られると言える．

今現在では，３Ｄ テクスチャマッピング機能は DeltaViewer に組み込まれ，実用化の段階にまで至っている．

付録

＜＜OpenGL のバージョン＞＞

OpenGL 1.2 以降では、ARB_imaging サブセットがオプションで使用できる環境もある。

OpenGL1.4

次の拡張機能が実装された.

・Blend　Squaring

RGBとアルファ値を２乗することを可能にするためにブレンド関数が拡張された。

従来拡張機能であった以下の機能が標準機能として実装された。

・generate_mipmap

拡張機能の GL_SGIS_generate_mipmap が標準機能になった。

・imaging

OpenGL 1.2, 1.3 でオプションであった ARB_imaging サブセットの glBlendEquation(), glBlendColor(),glBlendFunc の GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR, GL_CONSTANT_ALPHA, GL_ONE_MINUS_CONSTANT_CLPHA が標準機能となった。

・depth_texture

拡張機能の GL_ARB_depth_texture が標準機能となった。

・shadow

拡張機能の GL_ARB_shadow が標準機能となった。

・fog_coord

拡張機能の GL_EXT_fog_coord が標準機能となった。

・multi_draw_arrays

拡張機能の GL_EXT_multi_draw_arrays, (GL_SUN_multi_draw_arrays)が標準機能となった.

・point_parameters

拡張機能の GL_ARB_point_parameters が標準機能となった。

・secondary_color

拡張機能の GL_EXT_secondary_color が標準機能となった。

・blend_func_separate

拡張機能の GL_EXT_blend_func_separate が標準機能となった。

・

stencil_wrap

拡張機能の GL_EXT_stencil_wrap が標準機能となった。

・texture_env_crossbar

拡張機能の GL_ARB_texture_env_crossbar が標準機能となった。

・texture_lod_bias

拡張機能の GL_EXT_texture_lod_bias が標準機能となった。

・texture_mirrored_repeat

拡張機能の GL_ARB_texture_mirrored_repeat が標準機能となった。

・window_pos

拡張機能の GL_ARB_window_pos が標準機能となった。

OpenGL1.3

従来拡張機能であった以下の機能が標準機能として実装された。

・texture_compression

拡張機能の GL_ARB_texture_compression が標準機能となった。

・texture_cube_map

拡張機能の GL_ARB_texture_cube_map が標準機能となった。

・multisample

拡張機能の GL_ARB_multisample が標準機能となった。

・multitexture

拡張機能の GL_ARB_multitexture が標準機能となった。

・texture_env_add

拡張機能の GL_ARB_texture_env_add が標準機能となった.

・texture_env_combine

拡張機能の GL_ARB_texture_env_combine が標準機能となった。

・texture_env_dot3

拡張機能の GL_ARB_texture_env_dot3 が標準機能となった。

・texture_border_clamp

拡張機能の GL_ARB_texture_border_clamp が標準機能となった。

・transpose_matrix

拡張機能の GL_ARB_transpose_matrix が標準機能となった。

OpenGL1.2

従来拡張機能であった以下の機能が標準機能として実装された。

・**texture3D**
拡張機能の GL_EXT_texture3D が標準機能となった。glTexImage3D(), glTexSubImage3D(), glCopyTexSubImage3D() が追加された。

・**bgra**
拡張機能の GL_EXT_bgra が標準機能となった。
glDrawPixels(), glGetTexImage(), glReadPixels(), glTexImage1D(), glTexImage2D() が拡張された。

・**packed_pixels**
拡張機能の GL_EXT_packed_pixels が標準機能となった。

・**rescale_normal**
拡張機能の GL_EXT_rescale_normal が標準機能となった。
glEnable()/glDisable() が拡張され、GL_RESCALE_NORMAL を指定できるようになった。

・**separate_specular_color**
拡張機能の GL_EXT_separate_specular_color が標準機能となった。
glLightModel*() が拡張され、スペキュラカラーにテクスチャカラーの影響を受けない指定ができるようになった。

・**texture_edge_clamp**
拡張機能の GL_SGIS_texture_edge_clamp が標準機能となった。
glTexParameter*() の GL_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, GL_TEXTURE_WRAP_R に GL_CLAMP_TO_EDGE を指定できるようになった。

・**texture_lod**
拡張機能の GL_SGIS_texture_lod が標準機能となった。
glTexParameter*(), glGetTexParameter*() が拡張された。

・**draw_range_elements**
拡張機能の GL_EXT_draw_range_elements が標準機能となった。
glDrawRangeElements() が追加された。
次の ARB_imaging サブセット機能がオプションとして実装された。

・**blend_color**

拡張機能の GL_EXT_blend_color が ARB_imaging サブセットに含まれる。

・blend_minmax

拡張機能の GL_EXT_blend_minmax が ARB_imaging サブセットに含まれる。

・blend_subtract

拡張機能の GL_EXT_blend_subtract が ARB_imaging サブセットに含まれる。

・color_matrix

拡張機能の GL_SGI_color_matrix が ARB_imaging サブセットに含まれる。

・color_table

拡張機能の GL_SGI_color_table が ARB_imaging サブセットに含まれる。

・color_subtable

拡張機能の GL_EXT_color_subtable が ARB_imaging サブセットに含まれる。

・convolution

拡張機能の GL_EXT_convolution が ARB_imaging サブセットに含まれる。

・convolution_border_modes

拡張機能の GL_HP_convolution_border_modes (の大部分)が ARB_imaging サブセットに含まれる。

・histogram

拡張機能の GL_EXT_histogram が ARB_imaging サブセットに含まれる。

◆OpenGL1.1

従来拡張機能であった以下の機能が標準機能として実装された。

・vertex_array

拡張機能の GL_EXT_vertex_array が標準機能となった。

・polygon_offset

拡張機能の GL_EXT_polygon_offset が標準機能となった。

・blend_logic_op

拡張機能の GL_EXT_blend_logic_op が標準機能となった。

・texture

拡張機能の GL_EXT_texture が標準機能となった。

Texture Image Formats, Texture Replace Environment, Textire Proxies

・subtexture

拡張機能の GL_EXT_subtexture が標準機能となった。

・texture_objecxt

拡張機能の GL_EXT_texture_object が標準機能となった。

参考文献

[1] OpenGL Architecture Review Board
Manson Woo, Jackie Neider, Tom Davis:
OpenGL プログラミングガイド　（原著第２版）
OpenGL バージョン 1.1 対応オフィシャルガイド

[2] EDWARD ANGLE:
OpenGL A PRIMER

参考 ULR

[1] http://vivaldi.ics.nara-wu.ac.jp/~wada/DeltaViewer/index-j.html
[2] http://www.apple.co.jp/
[3] http://jp.nvidia.com/
[4] http://www.ati.com/jp/
[5] http://www.ccad.sccs.chukyou.ac.jp/manualc/prgrm/ANSI/
openGL/tex/index3.htm#TOP