2008年度　卒　業　論　文

# Gregory曲面を用いた滑らかさを保持した形状変形のGPGPUによる高速化手法

指導教員：渡辺 大地 講師

メディア学部 ゲームサイエンスプロジェクト
学籍番号　M0104258
武田　巧視

**2008 年度　　卒　業　論　文　概　要**

**論文題目**

# Gregory 曲面を用いた滑らかさを保持した形状変形の GPGPU による高速化手法


| | | | | |
|---|---|---|---|---|
| **メディア学部**<br>**学籍番号 : M0104258** | **氏名** | 武田　巧視 | **指導教員** | 渡辺 大地 講師 |



| | |
|---|---|
| **キーワード** | GPGPU, CUDA, OpenGL, 細分割曲面, Catmull-Clark 細分割,<br>パラメトリック曲面, Bézier 曲面, Gregory 曲面 |



3 次元コンピュータグラフィックス (以下、3DCG) における曲面表現に、数式によって表わすパラメトリック曲面と、ポリゴンと呼ぶ平面を組み合わせたポリゴンメッシュを利用するものとがある。パラメトリック曲面は滑らかな形状を正確に表すことが可能だが、描画速度や、複数の曲面を滑らかに繋げたままの動的な形状変形には向いていないという問題があり、リアルタイム 3DCG においては有用ではなかった。一方、ポリゴンメッシュは高速に形状を描画することが可能なため、リアルタイム 3DCG やキャラクターアニメーションの分野では需要が高まってきたが、形状が正確に表すことはできず、滑らかに表現するのも困難なので、設計などの分野では利用できない。

本研究では、画像処理専門のハードウェアである GPU を、汎用目的に利用する GPGPU という技術に着目し、パラメトリック曲面の描画に関する高速化手法を実現した。その際に、パラメトリック曲面の中でも、曲面同士を滑らかに繋げるという両立性補正が Bézier 曲面など、他のパラメトリック曲面に比べ単純な計算によって行える Gregory 曲面を用いた。結果、滑らかさを保持したままの動的な形状変形の高速化を可能にした。

# 目 次

# 第 1 章

# はじめに

3 次元コンピュータグラフィックス (以下、3DCG) における曲面表現に、数式によって表わすパラメトリック曲面と、ポリゴンと呼ぶ平面を組み合わせたポリゴンメッシュを利用するものとがある。パラメトリック曲面は滑らかな形状を厳密に表すことが可能だが、リアルタイムな描画を行うには処理が複雑であり、複数の曲面を滑らかに繋げたままの動的な形状変形にも複雑な計算が必要という問題があり、リアルタイム 3DCG においては有用ではなかった。一方、ポリゴンメッシュは曲面を厳密に表すことはできず、設計などの分野では利用できないが、高速に形状を描画することが可能なため、リアルタイム 3DCG や、キャラクタアニメーションの分野では需要が高まってきた [1][2]。ハードウェアの性能が向上し、シェーダと呼ぶグラフィックスハードウェア上で動作するプログラムが作成できるようになったため、3DCG における表現力が増した。しかし、従来の曲面表現では、用いる曲面式自体が複雑でリアルタイムな描画処理には向かず、動的な形状変形にも複雑な計算が必要になるという問題点がある。それらの問題点を解決するために、曲面計算を GPGPU を用いて高速に処理する研究が活発になっている [3][4]。GPGPU とは、画像処理を専門とする GPU を、グラフィックス描画という本来の目的ではなく、より一般的な計算用途に使う技術である。GPU は、トランジスタをデータ処理に振り向けており、1 つの GPU に計算処理ユニットが 100 以上も搭載していることもある。この計算処理ユニットをグラフィック描画のみなら

ず、他の数値演算にも利用するのがGPGPUのコンセプトである。このため、高度な並列計算を得意とし物理シミュレーションなど科学計算に向いているといえる。

本研究ではGPGPUを用い、既存研究とは別にパラメトリック曲面の一つ、Gregory曲面[5][6]を用いた。パラメトリック曲面は並列処理に向いた曲面表現であり、その中でもGregory曲面は曲面同士を単純な計算によって滑らかに繋げることができる曲面である。このGregory曲面を用い、GPGPUによって、滑らかさを保持したままの動的な形状変形を高速に描画することを実現した。パラメトリック曲面の動的な形状変形が高速に描画できるようになれば、高速な画面描画が必要なゲームやリアルタイムアニメーションにおいても、パラメトリック曲面が利用可能となる。

本論文の構成は以下のとおりである。第2章では、現在における3DCGの曲面表現から、本研究で用いるGregory曲面についての説明とGPGPUについて、解説する。第3章でGregory曲面の有用性とGPGPUを使用したときの検証と結果を示す。第4章で、本研究の成果と意義をまとめ、今後の展望について述べる。

# 第 2 章

# GPGPUと高速化に適した曲面表現について

本章では、曲面表現として利用するGregory曲面とは何か、なぜ高速化に適しているのかを、キャラクタアニメーションでの曲面表現手法として主流になっている細分割曲面と比較しながら明確化し、本研究で使用するGPGPUを解説する。2.1節では3DCGにおける曲面表現として、代数曲面、細分割曲面とパラメトリック曲面から実装に用いたGregory曲面を比較、2.2節はキャラクタアニメーション等でよく利用する細分割曲面とGregory曲面の比較、2.3節ではGPGPUと実装に用いたCUDAについて、2.4節でそれぞれの曲面とGPGPUとの相性について述べる。

## 2.1 3DCGにおける曲面表現

工業製品をつくるための滑らかな曲面を有する形状を生成するCAD(Computer Aided Design)やCAM(Computer Aided Manufacturing)、コンピュータグラフィックスでのモデルデータの設計など、形状に関する分野は幅広い。形状を扱う手法を整えることは大切な問題であり、表現手法の構築はその最も基礎となる部分である。曲面表現の一つに代数曲面がある。代数曲面では、円柱、球、円錐など、単純な形状表現には向いているが、表現できる形状が限定されるという特徴があ

る。代数曲面に比べ、より自由な曲面形状を表現する手法として有名なものに、ポリゴンメッシュとパラメトリック曲面がある。ポリゴンメッシュとは平面状の多角形を張り合わせたもので、滑らかではないが任意の形状を容易に近似できるという特性を持っている。部分的に線形になってしまうポリゴンメッシュの問題点を解決した手法が細分割曲面 (Subdivision Surfaces) である。次にパラメトリック曲面とは 2 パラメータの関数でもって曲面を表現する手法で、B-Spline 曲面や NURBS(Non-Uniform Rational B-Spline) 曲面、Bézier 曲面などがある。これらは滑らかな曲面を扱いやすいが、任意位相の曲面を関数で表現することは容易では無い。微妙な形状修正を行うために多数の制御点や高い次数の曲面を表す式が必要である。また、曲面間を滑らかに繋げるために相当に複雑な処理を行う必要がある。

### 2.1.1 代数曲面

代数曲面とは式 (2.1) を回転することで表現する曲面である。

$$Ax^2 + By^2 + Cz^2 = D \tag{2.1}$$

このとき、$A, B, C$ は実定数で、$A = B, C = 0$ ならば円柱を表し、$A = B = C > 0, D > 0$ ならば半径 $\sqrt{\frac{D}{A}}$ の球になる。同様に、$A = B, \frac{C}{A} < 0, D = 0$ ならば円錐を表す。代数曲面は単純な形状を表すことが可能だが、以下のような問題点がある。

- 限られた形状しか表すことができない。
- 係数と形状の関係が直感ではない。
- 式によって表わす形状の一部分を抜き出すことが困難である。

### 2.1.2 細分割曲面 (Subdivision Surfaces)

細分割曲面は単純な多面体形状に対して、分割と重み付けの操作を繰り返し適用することで、滑らかな曲面形状を生成する手法である。大まかにモデリングし

たポリゴンメッシュを細分化して、滑らかで継ぎ目の無い形状にする技術である。少ないポリゴン数で形状を滑らかに表現出来るため、編集や変形も容易になる。形状の作成時でも描画時でも、元のポリゴンメッシュに対する操作がリアルタイムに曲面に反映する。大量のデータの保存や調整をすることなく、高解像度のポリゴンメッシュの作成を行うことが可能であり、モデルに滑らかな生き物のような外観を与えることができる。

多面体形状を初期ポリゴンメッシュ、生成する曲面形状を極限曲面 (Limit Surface) と呼ぶ。Doo と Sabin[7] 及び Catmull と Clark[8] らが基礎理論を構築し、Loop[9] が三角形ポリゴンメッシュを対象とした手法を提案した。Doo と Sabin による手法 (Doo-Sabin 細分割) においては、極限曲面は双 2 次 B-Spline 曲面を近似する形状になる。Catmull と Clark による手法 (Catmull-Clark 細分割) においては、極限曲面は双 3 次 B-Spline 曲面を近似する形状になる。そのため、細分割曲面は B-Spline 曲面の一般化手法といえる。

Zorin ら [10] は Loop の細分割手法を局所的に適用することを可能にした。これにより、精度が不要な部分は荒く分割することが可能になり、細分割後の最終形状でもデータサイズをコンパクトにすることが可能になった。DeRose ら [11] は、Catmull と Clark の手法をアニメーション分野で現実的に応用することを試みた。彼らは複数の曲面形状をブレンドすることで、様々な曲面形状の作成を可能にしている。

Lee ら [12] は複雑な表面を持つ形状を、Loop 細分割曲面の初期ポリゴンメッシュとスカラー値の集合に分解して保存することにより大幅にデータを削減する手法を提案した。このように、細分割曲面は、任意の位相を持つポリゴン形状に適用することが可能であるため汎用性が高い。また、データサイズの小さい初期形状から複雑な曲面形状を生成することが可能である。一方で、最終的な生成形状を確認するためには複数の計算処理が必要である。また、既存の CAD/CG システムが認識可能な NURBS などの曲面制御点情報を明示的に保持していないという問題がある。

Catmull-Clark 細分割曲面は、任意位相の初期ポリゴンメッシュに細分割処理を繰り返し適用することで、滑らかな双 3 次 B-Spline 曲面に近似する極限曲面を生成する手法である。図 2.1 は、初期ポリゴンメッシュと、Catmull-Clark 細分割処理を 1 回適用したポリゴンメッシュ、2 回適用したポリゴンメッシュの例である。

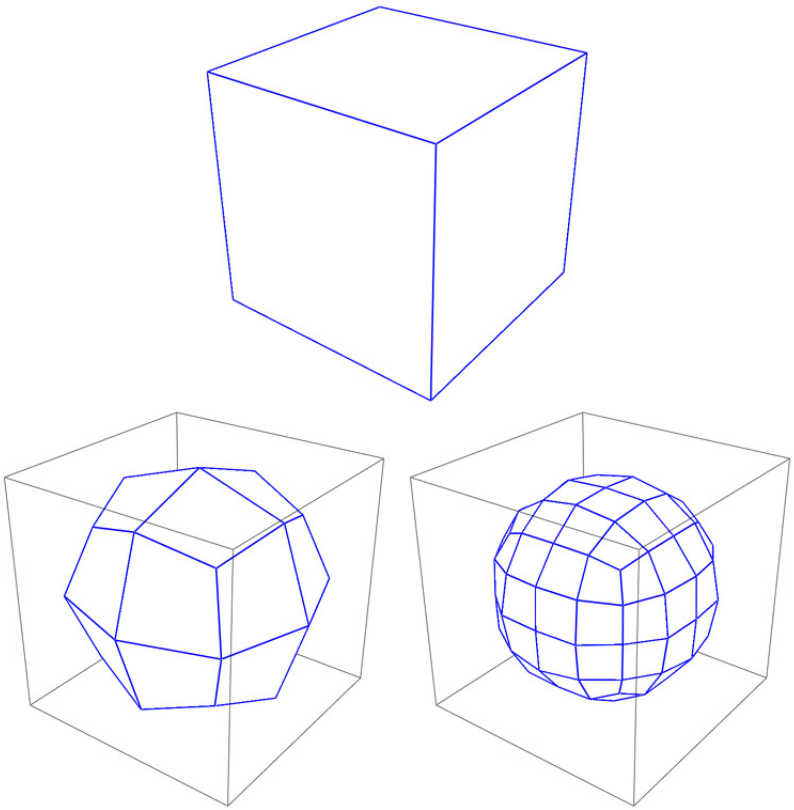

図 2.1: Catmull-Clark 細分割曲面

本研究では、パラメトリック曲面と比較する細分割曲面に Catmull-Clark 細分割曲面の分割手法を利用しているため、以下にその詳細を解説する。

分割により生成する新しいポリゴンメッシュの頂点は、分割前のポリゴンメッシュの頂点、辺、面に対応付ける。ここでは、分割前にあった頂点をメインポイント、辺を分割してできる頂点をエッヂポイント、エッヂポイントを繋げることで面上にできる頂点をフェイスポイントと呼ぶ。図 2.2 において、点線は分割前のポリゴンメッシュであり、実線は分割後のポリゴンメッシュである。

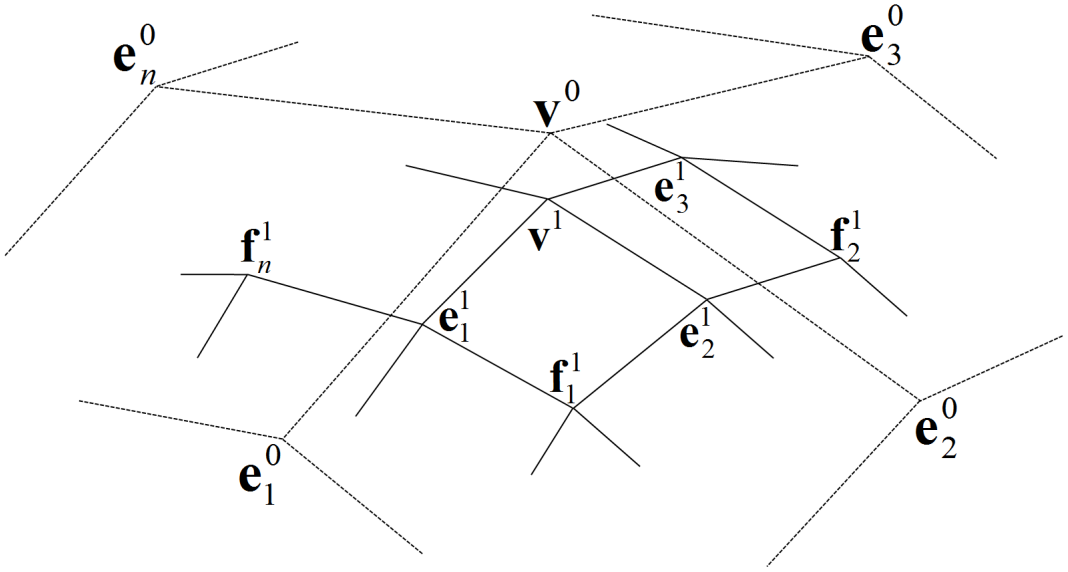


図 2.2: n 本の稜線と接続する頂点 $\mathbf{v}^0$ の周り

ポリゴンメッシュを分割した後、各頂点を再配置してポリゴンメッシュを滑らかにしていく。メインポイント $\mathbf{v}^0$ に接続している稜線の本数を $n$ とすると、分割後のポリゴンメッシュのフェイスポイント $(\mathbf{f}_1^1, \mathbf{f}_2^1, ..., \mathbf{f}_n^1)$ は分割前の各面の重心位置に生成する。式 (2.2) に示すように、エッヂポイント $(\mathbf{e}_1^1, \mathbf{e}_2^1, ..., \mathbf{e}_n^1)$ はその辺の両側のフェイスポイントと、両端のメインポイントの座標の平均とする。式 (2.3) に示すように、再配置後のメインポイント $\mathbf{v}^1$ は、再配置前のメインポイントの座標、分割前の形状でメインポイントにつながっている各稜線のもう一方のメインポイント、メインポイントの周りの各面のフェイスポイントの座標の平均とする。

$$\mathbf{e}_j^{i+1} = \frac{\mathbf{v}^i + \mathbf{e}_j^i + \mathbf{f}_{j-1}^{i+1} + \mathbf{f}_j^{i+1}}{4} \tag{2.2}$$

$$\mathbf{v}^{i+1} = \frac{n-2}{n}\mathbf{v}^i + \frac{1}{n^2}\sum_j \mathbf{e}_j^i + \frac{1}{n^2}\sum_j \mathbf{f}_j^{i+1} \tag{2.3}$$

### 2.1.3 パラメトリック曲面

パラメトリック曲面は、少数の制御点によって広域的で滑らかな形状を表現する手法である。代数曲面のように単純な数式では表わすことができず、空間に交点

と曲率をいくつか設定し、高次方程式でそれぞれの交点を補間して曲面を表現する。一般に多項式パラメトリック曲面を用いることが多く、デザインの分野を中心に広く利用している。Coons[13] は、計算機上でインタラクティブに曲面デザインを行う表現手法として Coons 曲面を開発した。Bézier[14] は、インタラクティブな自動車デザインのツールとして Bézier 曲面を開発した。4 つの曲線で囲まれた領域を持つ曲面をパッチと言い、Coons 曲面と Bézier 曲面の 2 つの曲面表現はともに、パッチを生成する手法である。これらの手法による曲面を表す式に、有理化による拡張や一般化を行うことにより、様々な曲面表現式の提案がある。Gordon と Riesenfeld[15] は、Bézier 曲面間の接続を滑らかにする B-Spline 曲面を提案した。B-Spline 曲面を有理化し、複数の曲面による 1 つの曲面表現を可能にした NURBS は、曲面作成の事実上の標準となっている [16]。パラメトリック曲面は厳密な形状の定義が可能であり、3 角形ポリゴンメッシュに比べてデータサイズも小さい。しかし、制御点を媒介とした変形操作には非常に熟練を要するため、意図した通りの変形が困難である。また、位相の変化を伴う変形を行った場合に、滑らかさを保つことも難しい。複数の曲面同士を滑らかに繋げることを両立性補正 [6] というが、Bézier 曲面や NURBS など、多くのパラメトリック曲面で必要とする両立性補正は非常に複雑な計算を必要とする。両立性補正の複雑さが、パラメトリック曲面の動的な変形を困難なものにしている要因の一つでもある。

### 2.1.4 Gregory 曲面

Gregory 曲面 [5][6] とは、Gregory による一般 Coons 曲面の拡張 [17] と同様の拡張を Bézier 曲面に対して施した曲面表現式である。

Bézier 曲面とは、制御点と呼ぶ位置ベクトルのみで曲面形状を定義するパラメトリック曲面の一種で、$n \times m$ 次の Bézier 曲面の曲面表現式は式 (2.4) のようになる。

$$\mathbf{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u) B_j^m(v) \mathbf{P}_{ij} \tag{2.4}$$

ただし、$0 \leqq u, v \leqq 1$ である。ここで、$\mathbf{P}_{ij}$ は Bézier 曲面の制御点を表す。制御点は $u$ 方向に $n+1$ 個、$v$ 方向に $m+1$ 個、合計 $(n+1)(m+1)$ 個ある。また、$B_i^n(u)$ と $B_j^m(v)$ は Bernstein 基底関数で式 (2.5) のように表す。

$$B_i^n(t) = \binom{n}{i} t^i (1 - \mathrm{t})^{n-i} \tag{2.5}$$

ここで、

$$\binom{n}{i} = {}_n\mathrm{C}_i = \frac{n!}{i!(n-i)!} \tag{2.6}$$

は 2 項係数である。また、Bézier 曲面の境界曲線は、Bézier 曲線となる。

このように Bézier 曲面による曲面形状は制御点を移動することで変更できる。しかし、隣り合う Bézier 曲面を滑らかに接続する場合には、非常に複雑な計算が必要になってしまうため、滑らかさを保ったままで形状を変更することは容易ではないという特徴がある。

この Bézier 曲面を拡張したものが Gregory 曲面である。双 3 次 Gregory 曲面は 20 個の制御点 $\mathbf{P}_{ijk}(i = 0, ..., 3; j = 0, ..., 3; k = 0, 1)$ で表現し、Gregory 曲面の曲面表現式は式 (2.7) のようになる。そのようすを図 2.3 に示す。

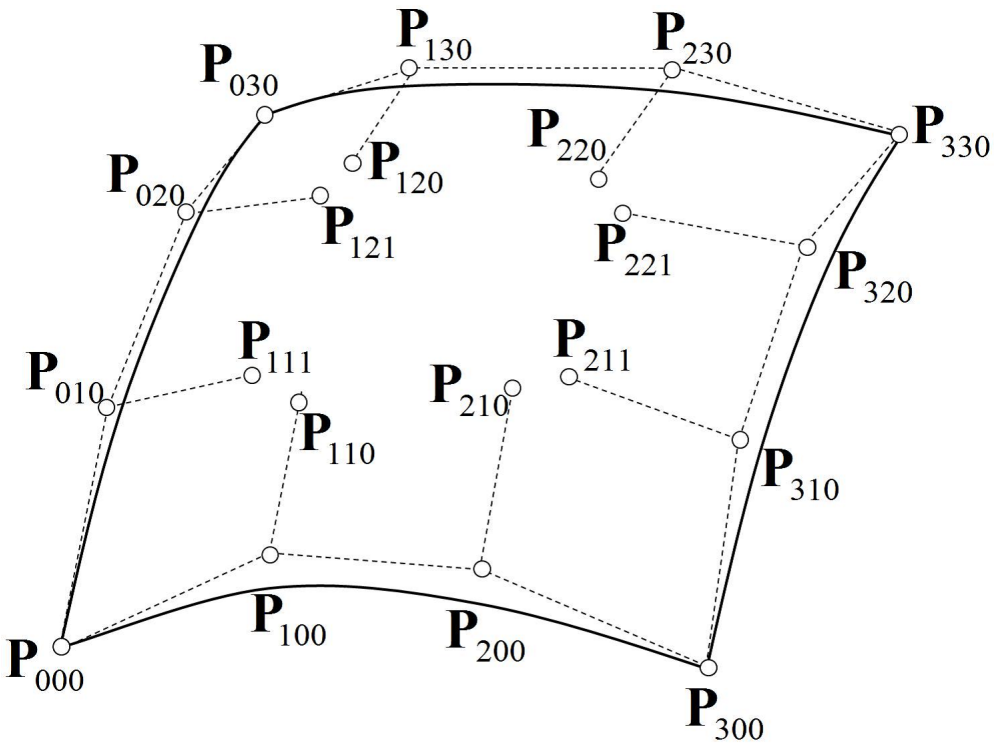


図 2.3: Gregory 曲面

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) B_j^m(v) \mathbf{Q}_{ij}(u,v) \tag{2.7}$$

ただし、$B_i^n(u)$ と $B_j^m(v)$ は式 (2.5) で表す、Bernstein 基底関数である。また、曲面の制御点 $\mathbf{P}_{ijk}$ と $\mathbf{Q}_{ij}$ には、次のような関係がある。

- $i \neq 1,2$ または $j \neq 1,2$ のとき

$$\mathbf{Q}_{ij} = \mathbf{P}_{ij} \tag{2.8}$$

- $i = 1,2$ かつ $j = 1,2$ のとき

$$\mathbf{Q}_{11}(u,v) = \frac{u\mathbf{P}_{110} + v\mathbf{P}_{111}}{u+v},$$

$$\mathbf{Q}_{12}(u,v) = \frac{u\mathbf{P}_{120} + (1-v)\mathbf{P}_{121}}{u+(1-v)},$$

$$\mathbf{Q}_{21}(u,v) = \frac{(1-u)\mathbf{P}_{210} + v\mathbf{P}_{211}}{(1-u)+v},$$
$$\mathbf{Q}_{22}(u,v) = \frac{(1-u)\mathbf{P}_{220} + (1-v)\mathbf{P}_{221}}{(1-u)+(1-v)} \tag{2.9}$$

ただし、$0 \leqq u, v \leqq 1$ である。

この曲面表現式は、以下のような特徴を持っている。

1. Gregory 曲面の境界曲線は Bézier 曲線になる。

2. Bézier 曲面と同様に凸閉包性を持つ。これは、曲面上の任意の点が、曲面の制御点で囲む領域の内部に存在するという性質である。この性質から、曲面間の干渉についてのラフなチェックを高速に行うことが可能である。

3. Gregory 曲面の特別な場合が、Bézier 曲面である。Gregory 曲面の制御点 $\mathbf{P}_{110}$ と $\mathbf{P}_{111}$、$\mathbf{P}_{120}$ と $\mathbf{P}_{121}$、$\mathbf{P}_{210}$ と $\mathbf{P}_{211}$、$\mathbf{P}_{220}$ と $\mathbf{P}_{221}$ がそれぞれ等しい時には、Gregory 曲面は Bézier 曲面と等しくなる。

さらに、境界線を共有する曲面同士の接続について述べる。図 2.4 は、接続する 2 枚の Gregory 曲面 $A(u,v), B(u,v)$ を表している。

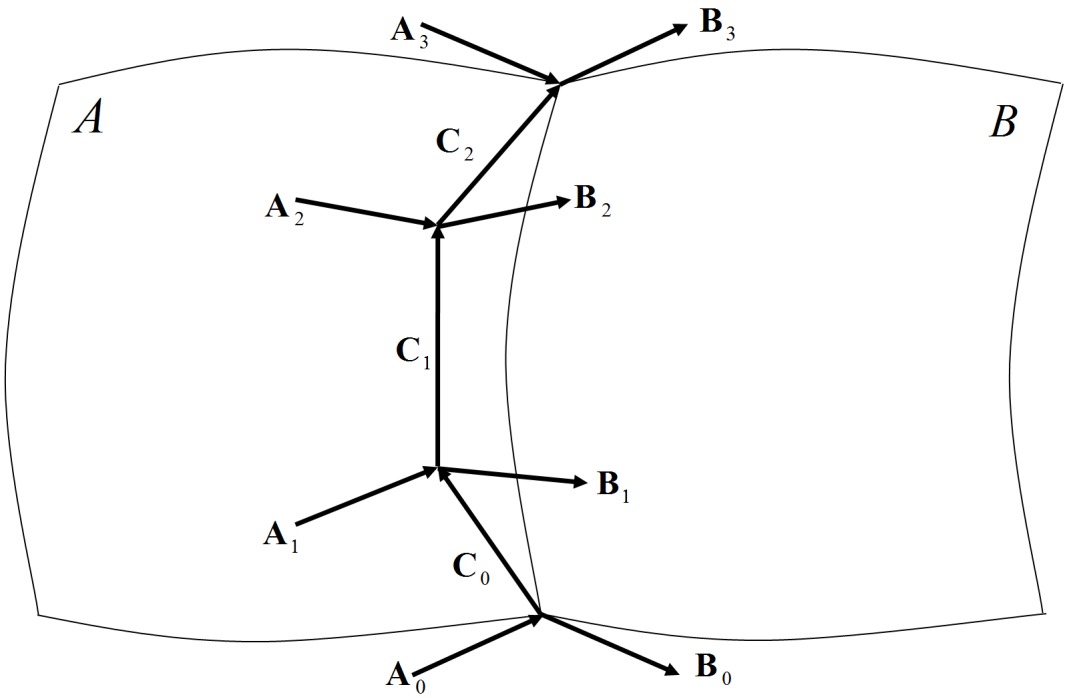


図 2.4: Gregory 曲面

ここで、境界曲線に隣接する制御点間ベクトルを $\mathbf{A}_i(i=0,...,3), \mathbf{B}_i(i=0,...,3), \mathbf{C}_i(i=0,1,2)$ とすると、

$$\mathbf{A}_1 = \frac{2\mathbf{A}_0 + \mathbf{A}_3}{3}, \mathbf{A}_2 = \frac{\mathbf{A}_0 + 2\mathbf{A}_3}{3} \tag{2.10}$$

と仮定したとき、式 (2.11) によって 2 つの曲面を滑らかに繋げるような $\mathbf{B}_1, \mathbf{B}_2$ を計算できる。

$$\begin{aligned} \mathbf{B}_1 &= \frac{(k_1 - k_0)\mathbf{A}_0 + 3k_0\mathbf{A}_1 + 2h_0\mathbf{C}_1 + h_1\mathbf{C}_0}{3}, \\ \mathbf{B}_2 &= \frac{3k_1\mathbf{A}_2 - (k_1 - k_0)\mathbf{A}_3 + h_0\mathbf{C}_2 + 2h_1\mathbf{C}_1}{3} \end{aligned} \tag{2.11}$$

ただし、$k_0, h_0, k_1, h_1$ は

$$\begin{aligned} \mathbf{B}_0 &= k_0\mathbf{A}_0 + h_0\mathbf{C}_0, \\ \mathbf{B}_3 &= k_1\mathbf{A}_3 + h_1\mathbf{C}_2 \end{aligned} \tag{2.12}$$

を満たす実数である。

隣り合う Bézier 曲面間を滑らかに繋げるためには、$u$ 方向と $v$ 方向同時に両立性補正を行う必要があり、一般的にこのことが形状を変形する上で大きな制約となった。一方 Gregory 曲面は $u$ 方向と $v$ 方向を独立に定義できる。Gregory 曲面同士を滑らかに接続する場合には、1 つの方向だけを意識すればよい。これは、曲面形状を生成するうえで大きなメリットとなる。単純な計算なので、Bézier 曲面などに比べて、両立性補正にかかる計算コストを非常に少なくできる。

## 2.2 細分割曲面と Gregory 曲面の比較

近年、曲面表現としては、パラメトリック曲面よりも細分割曲面が主役になろうとしている。その理由として、従来のパラメトリック曲面で複雑な曲面を表そうとすると、全体を 1 つの曲面で表すことができず、いくつものパッチと呼ぶ四辺形の曲面を貼り合わせて表現する必要があった。図 2.5 は 1 つのパラメトリック曲面で表せない形状を細分割曲面で表した例である。

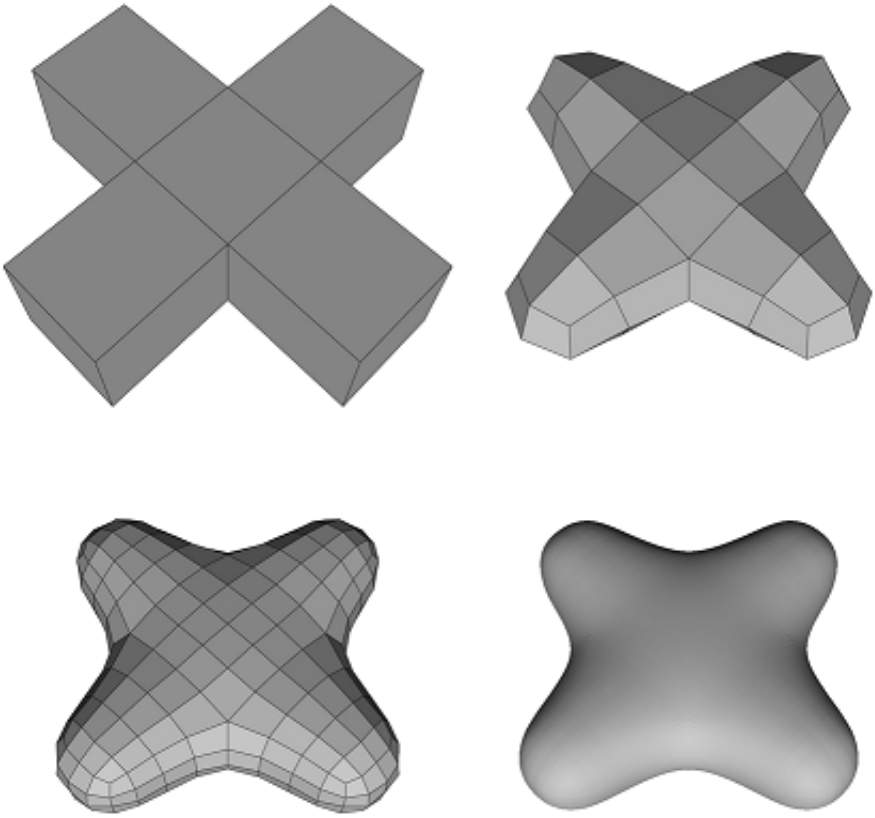

図 2.5: Catmull-Clark 細分割曲面の例

そのため、パッチとパッチの境界で滑らかに接続せずに折れたり、また、滑らかに接続するように曲面を生成しても、一部の曲面を変形すると、その滑らかさが壊れてしまい、再度調整が必要となる。それに対して、細分割曲面は全体を 1 つの曲面で覆うことができ、(いくつかの点を除いて) いたる所で滑らかな曲面を定義できるので、滑らかさを保ったまま自由自在に変形を行うことができる。パラメトリック曲面の中でも、Gregory 曲面は滑らかさを保ったままの動的な変形を、Bézier 曲面などに比べて簡単に行えるため、複数のパッチを使用した複雑な形状も表現しやすい。

細分割曲面は繰り返し分割処理を行うことでポリゴンメッシュを細かくしてい

くという特徴から、分割数の増加に伴って計算時間も増加していく。パラメトリック曲面の Bézier 曲面や Gregory 曲面では、どんな分割数であっても 1 度の計算で求めることができるため、分割数の違いによる計算時間の差異は細分割曲面に比べて少ない。

これらの特徴をまとめると表 2.1 のようになる。

表 2.1: 細分割曲面と Gregory 曲面の比較

| | 計算速度 | 形状変形 |
|---|---|---|
| 細分割曲面 | × | ○ |
| Bézier 曲面 | ○ | × |
| Gregory 曲面 | ○ | ○ |

## 2.3 GPGPU

近年、画像処理を専門とする補助演算装置 GPU(Graphics Processing Unit) の進化が、リアルタイム 3DCG の曲面表現に変化を見せ始めている。GPU を、画像処理という本来の目的ではなく、より一般的な計算用途に使う GPGPU(General Purpose GPU) の利用が広がりを見せてきた。GPU は、トランジスタをデータ処理に振り向けており、1 つの GPU に計算処理ユニットが 100 以上も搭載していることもある。このため、高度な並列計算を得意とし、物理シミュレーションなど科学計算に向いている。リアルタイムの画像処理は伝統的に、非常に高負荷な作業であるが、処理内容そのものは単純であるためハードウェア化に向いており、GPU は高速なメモリの搭載と強力な演算能力を集積した高性能化が著しい。CPU の処理能力が過去数年で数割ずつしか伸びていないのに対して、GPU の処理能力は年率 2 倍というスピードで進化している。用途や特性が違うため単純な比較は無意味だが、理論的な計算処理能力を比較すると、CPU が数 10G FLOPS であるのに

対して、GPU はすでに 1 チップで 500G FLOPS を超える製品がある。FLOPS とは、1 秒間に浮動小数点数演算 (実数計算) ができる回数である。90 年代中盤以降は 3D 描画性能が向上し、それに伴い行列演算を中心とした、1 回の命令で複数のデータに対する処理を同時に行う SIMD(Single Instruction Multiple Data) 演算機の必要性が増してきた。2000 年代に入ると、表現力の向上を求めて、プログラマが介入できない固定機能シェーダから、プログラマが任意に処理内容を記述できるプログラマブルシェーダへの移行が進み、演算の自由度が飛躍的に増した。高い処理能力の割に安価なグラフィックボードを使い、実際にシミュレーションなどを行う動きは、4 年ほど前に始まり、ここ 2 年ほど急速に普及の気配を見せ始めている。もともとは大学や研究機関の研究者たちが試行錯誤で GPU に並列計算を行わせていたが、こうした動きに目をとめた独立系 GPU ベンダの NVIDIA は、自社製品のハードウェアラインアップとして、2007 年 6 月に GPGPU 専用ボードやラックマウント型製品など「Tesla」シリーズをリリースした。同時に「CUDA」（Compute Unified Device Architecture）と呼ぶ開発環境の提供を始めた [18]。従来の GPU 向けプログラミング環境である Cg 言語は、CG 描画専用のプログラミング環境であり、汎用的なプログラムの記述が困難な環境であった。CUDA は標準の C 言語を採用しているため、汎用コンピューティングに適している。GPGPU が登場し、大量の並列演算が可能になったことで、種々の表現のさらなる向上が期待できるとともに、パラメトリック曲面などの複雑な計算を必要とする形状表現も高速に描画できるようになり、ここ数年研究が活発になっている [19]。本研究では GPGPU の実装を、この NVIDIA 社の CUDA を用いて行う。

## 2.4 GPGPU との相性

GPGPU との相性という点からみても、細分割曲面と Gregory 曲面には差がある。細分割曲面での繰り返し行う分割処理では、2 度目の分割処理には 1 度目の分割処理の結果を利用するので、計算の順番を考慮する必要がある。Catmull-Clark 細分割曲面の場合は 1 回の分割処理で、1 枚のポリゴンを $2 \times 2$ の 4 枚に分割する

ので、分割数が指数的な制限を受ける。細分割曲面はポリゴンメッシュを細かくしていくのが目的であり、分割数を増加させることはできるが、減少させることはできない。図 2.6 はポリゴンメッシュに細分割処理を 1 回行った場合、その結果を元に 2 回目の処理を行った例である。

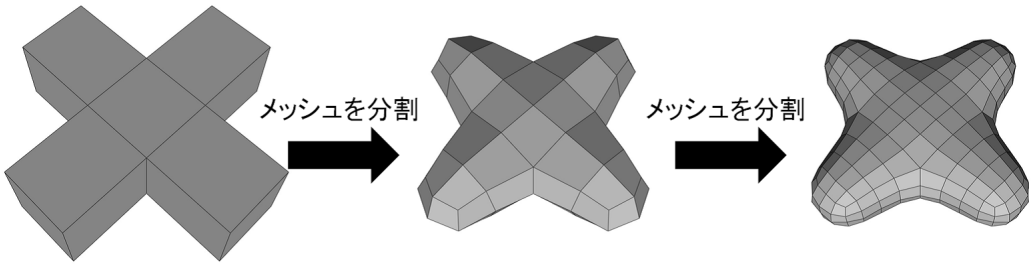


図 2.6: Catmull-Clark 細分割曲面の繰り返し処理

一方、Gregory 曲面や Bézier 曲面をポリゴンメッシュで表現する場合、数式によって求める曲面上の点を利用して面を生成することになる。利用するポリゴンメッシュの分割数の変更は、その増減や分割数に関わらず求めることが可能である。図 2.7 は、$5 \times 5$ 分割のポリゴンメッシュで表した Gregory 曲面を、$3 \times 3$ 分割、$10 \times 10$ 分割に変更した例である。

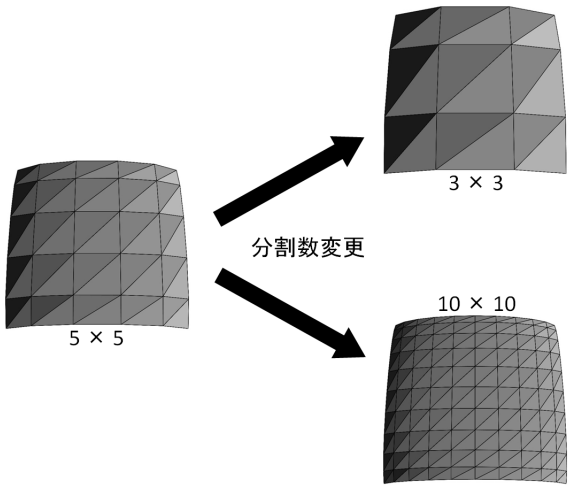


図 2.7: Gregory 曲面は分割数が違っても 1 度で計算できる

Gregory 曲面の曲面式によって求める点は個々に関連性がないため、計算する順番などは意識する必要はない。

細分割曲面では 1 度の分割処理に関しても個々の頂点が関連し合い、分割処理も繰り返し行わなければならないため、GPGPU のような並列計算に向いているとは言えない。対して、Gregory 曲面は 1 度の計算によって頂点が求められ、個々の頂点に関連性がないことから、並列計算との相性が優れているといえる。

表 2.1 に GPGPU との相性を含めると表 2.2 のようになる。

表 2.2: GPGPU との相性を含む細分割曲面と Gregory 曲面の比較

| | 計算速度 | 形状変形 | 並列計算 |
|---|---|---|---|
| 細分割曲面 | × | ○ | × |
| Bézier 曲面 | ○ | × | ○ |
| Gregory 曲面 | ○ | ○ | ○ |

これらのことから、本研究では、滑らかさを保ったまま高速に形状変形でき、並列計算に向いている曲面表現として Gregory 曲面を採用する。

# 第 3 章

# 検証

本章では、GPGPU を用いて Gregory 曲面の、滑らかさを保持した形状変形の高速化を実装したプログラムを使用し、その有用性を検証する。このプログラムは、グラフィックス API の OpenGL[20] と OpenGL をベースとした 3DCG ツールキットである FK Toolkit System[21] 、GUI ライブラリである Lily Libraly[22]、GPGPU には NVIDIA 社の CUDA を用いて実装した。検証に用いた環境は以下の通りである。

| | |
|---|---|
| CPU： | Intel(R) Core(TM)2 Duo 3.00GHz |
| メインメモリ： | 2.00GB RAM |
| GPU： | NVIDIA GeForce 9600 GT |
| 解像度： | SXGA (1280 × 1024) |

## 3.1 結果

### 3.1.1 Gregory 曲面の両立性補正

Gregory 曲面の両立性補正処理の有用性を図るために、両立性補正処理を行った場合と、行わなかった場合で、形状に変形を加えたときの描画速度を比較した。$2 \times 3$ パッチで旗がはためいているように見えるように稜線をアニメーションさせ、Gregory 曲面を内挿する。Gregory 曲面は $10 \times 10$ 分割のポリゴンメッシュで表現した。結果、速度にほとんど変化はなく、Gregory 曲面では両立性補正において速度

低下は起きないといえる結果になった。10 秒間形状変形を続けたときの FPS(秒間フレーム数) の平均を表 3.1 にまとめた。図 3.1 は定義した稜線と、稜線に Gregory 曲面を内挿したものである。

表 3.1: Gregory 曲面の両立性補正の速度比較

| | |
|---|---|
| 両立性補正あり | 185.3FPS |
| 両立性補正なし | 185.9FPS |

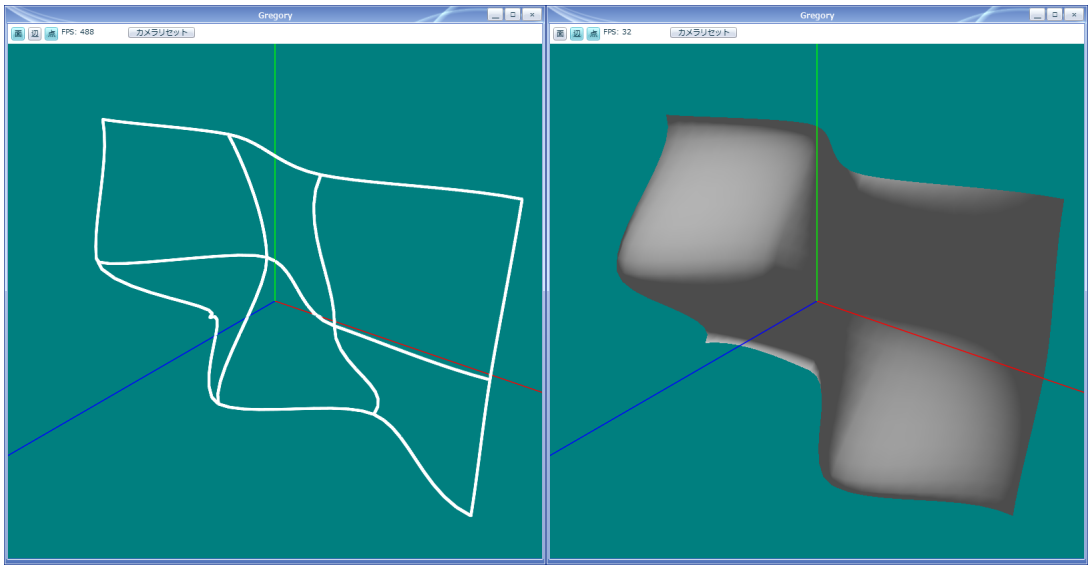


図 3.1: 両立性補正実験

### 3.1.2 細分割曲面と Gregory 曲面

GPGPU を利用しない状態での細分割曲面と Gregory 曲面の速度比較を行った。図 3.2 は、$12 \times 12$ 分割のポリゴンメッシュで表現した Gregory 曲面と細分割曲面の例である。

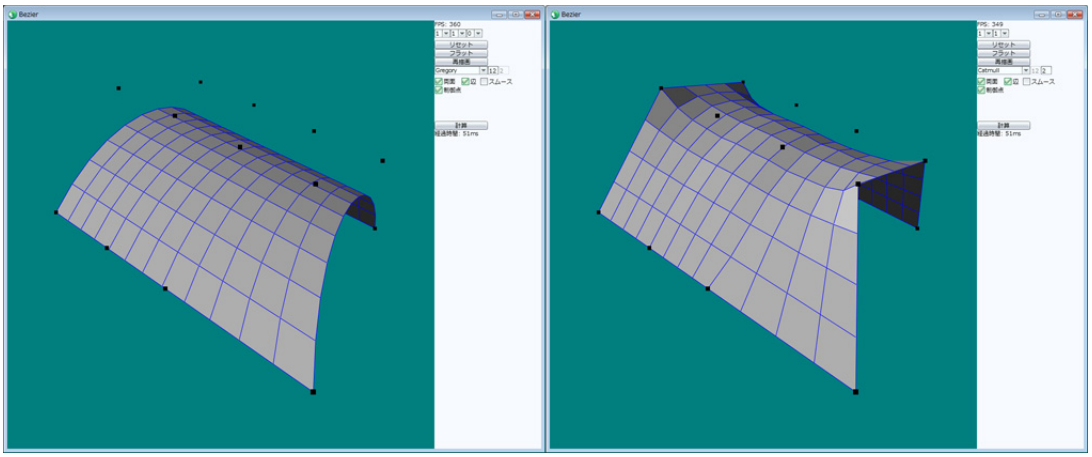

図 3.2: Gregory 曲面と細分割曲面の比較実験

比較方法は、曲面生成処理を 10 回繰り返した際にかかる時間を、分割数を変えて比較した。3 × 3 の四角形ポリゴンメッシュに対して細分割処理を行い、その時に生成した分割数に Gregroy 曲面の分割数を合わせた。10 回計測を行った時の平均値は表 3.2 のようになる。

表 3.2: 細分割曲面と Gregory 曲面の速度比較

| 分割数 | 6 × 6 | 12 × 12 | 24 × 24 | 48 × 48 | 96 × 96 |
|---|---|---|---|---|---|
| 細分割曲面 | 8.7 ms | 43.9 ms | 237.6 ms | 1489.6 ms | 13208.6 ms |
| Gregory 曲面 | 3.2 ms | 18.1 ms | 106.8 ms | 744 ms | 6930.1 ms |

このように、全体的に Gregory 曲面が速くなるという結果が得られた。

### 3.1.3 Gregory 曲面と GPGPU

Gregory 曲面はポリゴンメッシュで表現するため、面を張るための頂点配列を求める部分に GPGPU を利用した。ハードウェアの性能上、GPU 側のメモリから CPU 側のメモリへのデータ転送が発生すると、オーバーヘッドになり速度低下につながる。OpenGL には VBO(Vertex Buffer Object) と呼ばれる、GPU 側のメ

モリ上にデータを生成し、描画の際に利用する機能があり、CUDA も対応している。GPU 側のプログラムに Gregory 曲面の制御点情報を転送し、求めた頂点情報を VBO に格納することで、オーバーヘッドを抑えることが可能になる。

このプログラムにおいて、曲面の生成、及び変形後の再描画にかかる時間を計測した。図 3.3 は平面上に配置した 6 枚のパッチの形状変形前と変形後の様子である。

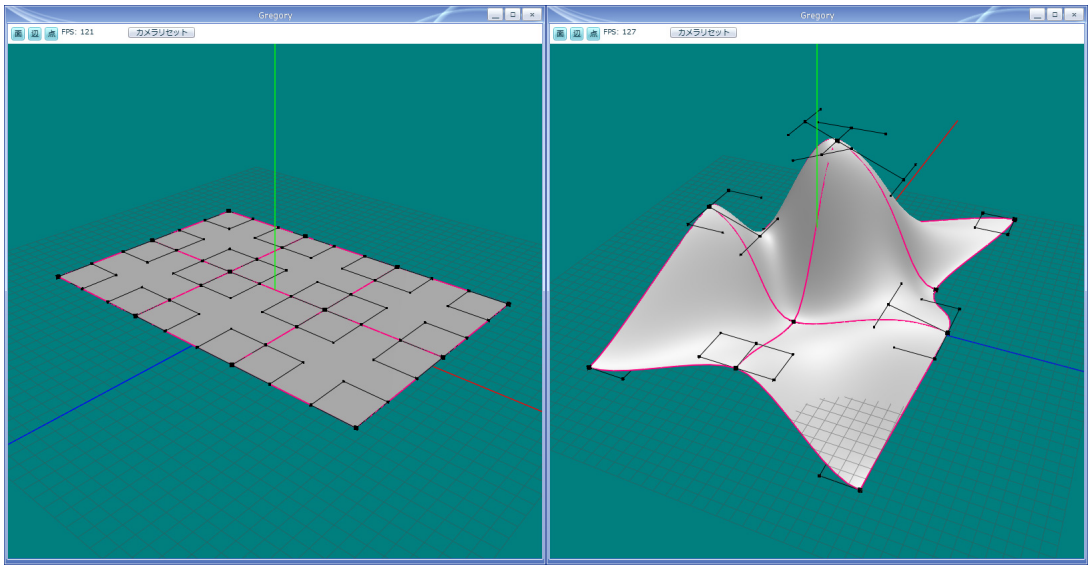


図 3.3: Gregory 曲面の形状変形前と変形後

GPGPU を利用しないプログラムでは、1 パッチの生成に 30〜60 ミリ秒かかったが、GPGPU を利用した場合、ほぼ 1 ミリ秒以下の速度となった。しかし、別の環境で計測したところ GPGPU を利用しない方が速度が速いという結果になった。利用した環境の構成は、以下のとおりである。

| | |
|---|---|
| CPU： | Intel(R) Core(TM)2 Duo 2.40GHz |
| メインメモリ： | 4.00GB RAM |
| GPU： | NUVIDIA GeForce 8400M GT |
| 解像度： | SXGA (1280 × 1024) |

GPGPU の使用で高速化できた環境を A、高速化できなかった環境を B として、曲面を動的に変形し、再描画にかかった時間 10 回分の平均値を表 3.3 にまとめる。

表 3.3: 細分割曲面と Gregory 曲面の速度比較

| | | |
|---|---|---|
| A | CPU のみ | 44.0784248 ms |
| A | GPU 利用 | 0.3748625 ms |
| B | CPU のみ | 53.5556265 ms |
| B | GPU 利用 | 378.1104234 ms |

このことから、GPGPU を利用することで、曲面の動的な変形の高速化が実現できたことが分かるが、速度がハードウェアの性能に依存するという結果になった。

## 3.2 問題点

現状の問題点として、Gregory 曲面を 1 ピクセル以下の大きさしかないマイクロポリゴンによるポリゴンメッシュで表現しているが、分割数の動的な増減には対応していない問題が挙げられる。曲面に変更を加えたり、カメラが移動する際に必要なマイクロポリゴンの大きさを求め、分割数を変えて曲面を描画し直すことで対処が可能であると考える。また、CUDA の仕様として、大量のデータを保持できない、個々のハードウェアによる性能の差が著しい等があげられる。この仕様のために、現在の手法では実現できない処理などの対処も必要になる。

# 第 4 章

# おわりに

本研究では、Gregory 曲面を用いた滑らかさを保持した形状変形の GPGPU による高速化手法を提案し、3DCG における曲面表現の高速化手法の研究に取り組んだ。その成果として、従来の手法よりも、曲面の動的な変形を高速に行うことができた。また本手法は、複数の曲面を滑らかに繋げたままの変形が可能である。本手法を用いることで、生き物のような滑らかな形状モデルの動的な変形を実現できるため、ゲーム等リアルタイム 3DCG コンテンツを制作する際の表現力の向上に役に立つだろう。

今後の展望として、3.2 節で挙げた分割数の動的な変化の問題の対処や高速化は行えたが、リアルタイム処理を行えるほどの速度は出ていないため、さらなる高速化手法の提案が必要になるだろう。限られたグラフィックメモリを使用しているため、パッチを動的に増減させたときの速度変化についても対処が必要だろう。

# 謝辞

本研究を締めくくるにあたり、研究の指針から開発の手法、論文の執筆と幅広いご指導ご教授を頂きました、本校メディア学部の渡辺大地講師、並びに三上浩司講師に厚く感謝いたします。また、プログラムの実装にあたり、すばらしいライブラリを開発し、当時公開されていなかったにもかかわらず使用する許可を下さった、本校メディア学部の嘱託研究員、渡辺賢悟氏に厚く感謝いたします。

そして、様々な相談に乗ってくださった先輩方、多くの苦楽を共にしたゲームサイエンスプロジェクトの仲間たちに感謝いたします。特に我々と共に、泊り込んでまで添削をしてくださった、阿部雅樹先輩には心より感謝いたします。

我唯一唯我也。何は無くとも地球は廻る。

# 参考文献


[1] 脇田玲, 矢島誠, 原田毅士, 鳥谷浩志, 千代倉弘明, ”ラティス構造に基づく軽量で高品質な Web3D データ表現”, 情報処理学会論文誌, 第 42 巻 第 5 号, 2001.

[2] 鈴木宏正, ”自由形状を表現するサブディビジョンサーフェスモデリング”, 日本機械学会誌　2001. 4 Vol. 104 No. 989.

[3] Charles Loop, Jim Blinn, ”Real-Time GPU Rendering of Piecewise Algebraic Surfaces”, SIGGRAPH, 2006.

[4] Juraj Konečný, ”Catmull_Clark Subdivision Surfaces on GPU”, CESCG, 2007.

[5] H. Chiyokura and F. Kimura. ”Design of solids with free-form surfaces”. Computer Graphics, 17(3):289-298, 1983.

[6] 鳥谷 浩志, 千代倉 弘明, 「3 次元 CAD の基礎と応用」, 1991.

[7] D. Doo and M. Sabin, ”Behaviour of recursive division surfaces near extraordinary points”, Computer Aided Desin, 1978.

[8] E. Catmull and J. Clark, ”Recursively generated b-spline surfaces on arbitrary topological meshes”, Computer Aided Design, pp.350-355, 1978.

[9] C. Loop, "Smooth subdivision surfaces based on triangles", Master's thesis, University of Utah, Department of Mathematics, 1987.

[10] D. Zorin, P. Schroder and W. Sweldens, "Interactive multiresolution mesh editing", Computer Graphics (Proc. SIGGRAPH 97), pp.259-268, ACM Press, New York, 1997.

[11] T. DeRose, M. Kass, T. Truong, "Subdivision surfaces in character animation", Computer Graphics (Proc. SIGGRAPH 98), pp.85-94, ACM Press, 1998.

[12] A. Lee, H. Moreton and H. Hoppe, "Displaced subdivision surfaces", (SIGGRAPH'98 Proceedings), pp.95-104, 1998.

[13] S. A. Coons, "Surfaces for computer-aided design of space figures", MIT, 1964.

[14] P. Bézier, "Definition numerique des courbes ed surfaces", Automatisme, 1996.

[15] W. J. Gordon and R. F. Riesenfeld, "Bernstein-Bézier methods for the computer aided design of free-from curves and surfaces", journal of the ACM, 21:293-310, 1974.

[16] G. Farin, "Curves and surfaces for computer aided design", Academic Press, 1992.

[17] J. A. Gregory, "Smooth interpolation without twist constrants", Computer Aided Geometric Design, R. E. Barnhill and R. F. Riesenfeld, ed., Academic Press, 1974.

[18] CUDA, <http://www.nvidia.co.jp/object/cuda_home_jp.html>.

[19] Charles Loop, Jim Blinn, "Resolution Independent Curve Rendering using Programmable Graphics Hardware", SIGGRAPH, 2005.

[20] OpenGL.org, OpenGL, <http://www.opengl.org/>.

[21] 渡辺大地, FK Tool Kit System , <http://fktoolkit.sourceforge.jp/>.

[22] 渡辺賢悟, Lily Library , <http://kengolab.net/>.