修士論文　　2020 年度

テッセレーションシェーダを用いた
描画要素数の動的制御による地形描画高速化に関する研究

東京工科大学大学院

バイオ・情報メディア研究科

メディアサイエンス専攻

山本 馨加

修士論文　　2020 年度

テッセレーションシェーダを用いた
描画要素数の動的制御による地形描画高速化に関する研究

指導教員　　渡辺 大地 教授

東京工科大学大学院

バイオ・情報メディア研究科

メディアサイエンス専攻

山本 馨加

論 文 の 要 旨

論文題目	テッセレーションシェーダを用いた 描画要素数の動的制御による地形描画高速化に関する研究
執筆者氏名	山本 馨加
指導教員	渡辺 大地 教授
キーワード	Tessellation Shader、Gregory 曲面、地形生成、GPU、パーリンノイズ

[**要旨**]

近年、PC や家庭用ゲーム機などのハードウェアの性能が上がり、広大な地形を移動して遊ぶことができるオープンワールドゲームが多く存在している。しかし、多くの 3D モデルや高精細に広大な地形を描画すると膨大な計算処理が必要となってしまう。そのため、カメラとの距離が近い位置のモデルは多くのポリゴン数を使い高精細に描画を行い、遠くのモデルはポリゴン数を抑えて表示を行う LevelOfDetail という方法を使用して計算処理を軽減している。一般的に GPU を使って 3D モデルを描画する場合、始めに一度だけモデルの情報を GPU に送りメモリ上に保存する。その後、メモリ上に保存したモデル情報を使い描画を行う。そして、表示するモデルの頂点数やポリゴンを構成するトポロジーの情報に変更があった場合、変更のある部分の情報の再送信を行う。LevelofDetail のように、ポリゴン数が大きく変わる場合、切り替える 3D モデルの全情報を再送信しなければならない。CPU から GPU へ多くの情報を送信してしまうと処理がおもくなるため、情報の送信はなるべく減らす必要がある。本論文では、パラメトリック曲面の一つである Gregory 曲面とテッセレーションシェーダとパーリンノイズを使い、再送信する情報量の少ない地形の生成方法を実現した。各曲面の領域情報と制御点情報を GPU に一度だけ送り、カメラの距離や位置情報をもとに生成する地形の範囲を計算し送信することで、再送信の情報量が少ない地形を生成を行った。広い範囲の地形生成を行った場合から、いくら狭い範囲の地形の生成を行った場合でも送信する情報量は変わることがない。また、周波数や振幅の違う複数パーリンノイズを合成して地形の高さ方向に使用して、生成する地形の範囲に対応した形状の表示も行うことができた。そして、フラグメントシェーダで地形の高さ方向とパーリンノイズを使用して色付けを行った。

A b s t r a c t

Title	A Study on Speeding Up Terrain Drawing by Dynamically Controlling the Number of Drawing Elements Using a Tessellation Shader
Author	Kiyoka Yamamoto
Advisor	Taichi Watanabe
Key Words	Tessellation Shader, GregorySurface, Terrain Generation,GPU, Perlin Noise,

[**summary**]

In recent years, the performance of hardware such as PC and home video game consoles has increased, and there are many open-world games that allow players to move through vast terrains. However, rendering a large number of 3D models and vast terrains in high definition requires an enormous amount of computational processing. For this reason, the LevelofDetail method is used to reduce the computation process by using a large number of polygons to draw models at a close distance from the camera in high definition, and a smaller number of polygons to display models at a distance. In general, when using a GPU to draw a 3D model, the model information is first sent to the GPU only once and saved in memory. After that, the model information stored in the memory is used to draw the model. When the number of vertices or the topology of polygons of the model to be displayed changes, the information of the changed part is resent. In this paper, we propose a new method of sending information from the CPU to the GPU. In this paper, we use Gregory surfaces (a parametric surface), tessellation shaders, and Perlin noise to generate terrain with less information to be retransmitted. By sending the area information and control point information of each surface to the GPU only once, and then calculating and sending the range of the terrain to be generated based on the distance and position information of the camera, we were able to generate terrain with less information to be retransmitted. The amount of information to be transmitted does not change no matter how narrow the terrain is generated from a wide range. We were also able to synthesize multiple perlin noises with different frequencies and amplitudes and use them in the height direction of the terrain to display the shape corresponding to the range of terrain to be generated. The fragment shader then used the terrain height direction and the perlin noise for coloring.

目 次

第 1 章　はじめに 1
1.1　研究背景 2
1.2　論文構成 6

第 2 章　提案手法 7
2.1　提案手法の方針 8
2.2　パラメトリック曲面 10
2.3　Gregory 曲面 10
2.4　各曲面の領域情報 13
2.5　生成する地形の範囲情報 14
2.6　カメラ情報をもとにした生成する地形の範囲情報の算出 15
2.7　CPU 側での地形の生成の準備 20
2.8　GPU 側での地形の生成 21

第 3 章　出力結果と検証 26
3.1　開発環境 27
3.2　出力結果 27
3.3　本手法と従来手法の比較 35
3.4　結論と考察 36

第 4 章　まとめ 37

謝辞 39

参考文献 42

発表業績 46

図 目 次

2.1 複数枚の曲面をつなぎ合わせたイメージ 8
2.2 曲面のみの形とノイズを加えた形のイメージ 9
2.3 各の曲面の最小値と最大値の設定 14
2.4 生成する地形の範囲の設定 15
2.5 地形全体を一定間隔で区切った例 16
2.6 カメラの描画範囲にある点をもとに最小値と最大値を決めた例 19
2.7 カメラの描画範囲にある点をもとに最小値と最大値を決めた例 20
2.8 テッセレーションシェーダによる新たな形状生成例 22
2.9 送信した情報をもとに曲面を生成した場合の頂点の例 23

3.1 g と h が 1.0 の地形 28
3.2 g と h が 0.95 の地形 29
3.3 g と h が 0.9 の地形 29
3.4 g と h が 0.85 の地形 29
3.5 g と h が 0.8 の地形 29
3.6 g と h が 0.75 の地形 29
3.7 g と h が 0.7 の地形 29
3.8 g と h が 0.65 の地形 30
3.9 g と h が 0.6 の地形 30
3.10 g と h が 0.55 の地形 30
3.11 g と h が 0.5 の地形 30
3.12 g と h が 0.45 の地形 30
3.13 g と h が 0.4 の地形 30
3.14 g と h が 0.35 の地形 31
3.15 g と h が 0.3 の地形 31
3.16 g と h が 0.25 の地形 31

3.17 g と h が 0.2 の地形 31
3.18 g と h が 0.15 の地形 31
3.19 g と h が 0.1 の地形 32
3.20 g と h が 1.0 の地形とワイヤーフレームの表示 33
3.21 g と h が 0.5 の地形とワイヤーフレームの表示 33
3.22 g と h が 0.2 の地形とワイヤーフレームの表示 34
3.23 g と h が 0.05 の地形とワイヤーフレームの表示 34
3.24 g と h が 0.005 の地形とワイヤーフレームの表示 35

第 1 章

はじめに

1.1 研究背景

近年、PC や家庭用ゲーム機などのハードウェアの性能が上がり、多くの処理が可能となった。ハードウェア性能の向上により、ゲームでは画面内に多くの建物やキャラクターなどの 3D モデルを表示することができる。FINALFANTASY15[1] や FARCRY5[2] や Ghost of Tsushima[3] などのオープンワールドゲームと呼ばれる、広大な地形を移動して遊ぶことができる作品が多く制作されている。しかし、ゲームはリアルタイムに処理を行い、快適なプレイの実現が必要な分野である。ハードウェアの性能が向上し、多くの処理ができるようになったが、広大な地形を全て高精細に描画を行ったり、ポリゴン数の多い 3D モデルを多く表示してしまうと、計算処理が多くなり快適なプレイが実現できなくなってしまう。そのため、表示するの地形やキャラクターの表示数やポリゴン数を調整する必要がある。ゲームでは計算処理を減らす方法として 3D モデルの詳細度制御を行っている。一般的に Level of Detail（以降「LOD」）と呼ばれる方法であり、事前にポリゴン数の違う 3D モデルを複数体用意しておき、カメラと 3D モデルの距離に合わせて表示するモデルを切り替えていく。カメラから離れた位置にある 3D モデルはポリゴン数を抑えたものを使い表示を行い、カメラから近い位置にある 3D モデルにポリゴン数が多く詳細なものを使うことで、画面の見た目を大きく損なわずに計算処理を軽くしている。

GPU[4][5] を使い 3D モデルを表示する場合、GPU に 3D モデルの情報を一度だけ送り込み、GPU 内のメモリ上に送信した情報を保存しておく。その後、GPU のメモリ上に保存した 3D モデルの情報を使い、表示を行っていく。また、表示するモデルの頂点数やポリゴンを構成するトポロジーに変更がある場合、変更の起こった部分の情報を新たに GPU に送信しなければならない。そして、多くの情報を CPU から GPU へ送信してしまうと処理が重くなってしまうため、再送信する情報は減らす必要がある。LOD ではカメラと表示する 3D モデルの距離に応じて、表示する 3D モデルを切り替えている。そのため、カメラとモデルの距離が変わり、表示する 3D モデ

ルの切り替えが発生した場合、GPU に新しい 3D モデルの情報の送信する必要がある。LOD を使いポリゴン数を抑える場合、事前に GPU のメモリ上に保存していた情報と比べ、頂点数やトポロジーが大きく変化するため、モデルの情報を全て再送信することになる。

オープンワールドゲームなどの広大な地形を生成する場合、描画処理を行う前に広大な地形をいくつかの段階にわけて分割して CPU 上で準備しておき、カメラとの距離に応じて分割した地形を GPU へと送信して表示を行ってる。FARCRY5[6] では、最初に地形生成に必要な形状データであるハイトマップやノーマルマップなどを何段階かのミップマップとして準備する。次に描画処理を始める前に CPU 内で何キロもの大きな地形を繰り返し分割していき、分割するたびに分割した地形情報の保存を行い、複数の段階で分割した地形を保存しておく。次に広大な地形のどこにカメラがいるのか位置の算出し、先ほど分割し保存しおいた情報から適切な地形を選んでいく。カメラ位置を元に、カメラと地形の距離が遠い時には分割回数が少なく一つ一つが大きい地形を選び、カメラ地形の距離が近い時には分割回数が多く一つ一つが小さい地形を選ぶ。分割した地形と選んだ分割段階に対応する最初に準備した地形生成に必要な形状データを GPU に送る。そして、GPU に送られた情報をもとに、どの分割段階の地形を選択し、カメラに映らない部分の地形の描画処理を無視する。最終的に送られた情報をもとに地形を生成することでカメラ距離に応じた描画処理を行っている。しかし、この方法ではより細かな地形を表示しようとした場合、さらに細かい分割を行いその段階の情報を保持しておき、それに対応する地形生成に必要な形状データの準備が必要となる。そのため、準備する情報量も増えてしまい、表示する地形が細かくなればなるほど GPU への多くの情報の送信が必要となってしまう。

地形生成に関する研究は昔から多く行われており、非整数ブラウン運動を用いた生成方法 [7] や中点変位法を用いた手法 [8] などのフラクタル手法を使ったものがある。また、等高線や標高値などから地形モデル生成する手法 [9] や等高線と B-spline 曲線を用いた生成手法 [10] のように、事前に用意した情報から地形を生成する方法も存在する。そして、ハードウェアの性能が向上し

GPU を用いた地形の生成手法が研究され考えられるようになり、GPU を用いた地形生成として Losasso ら [11] の研究や Hwa ら [12] の研究などが存在する。また、四分木構造というデータの構造を用いた生成手法の研究が考えられるようになり Zhang ら [13] の研究や四分木構造と GPU のテッセレーションを使った Strugar[14] の研究、Strugar の研究を改良した Judnich ら [15] の研究などが存在する。近年では、GPU を使ったリアルタイムの地形レンダリングの方法として、2 つの四分木構造を使った Rui ら [16] の研究があるが、事前に地形を分割して保持しておき、生成する地形にあわせて事前に分割しておいた情報を GPU に送ることで細かな地形の表示を実現している。そのため、カメラの位置が変わり生成したい地形が変わると多くの情報を再送信する。また、広大な地形の細かな形状を生成する研究として HyeongYeop ら [17] の研究があり、カメラ距離に合わせて地形を生成して、ノイズを使用し距離に応じた形状を表現することができる。しかし、彼らの研究でも事前に地形を分割して保持しておき、必要な分割した地形と形状データを GPU に送ることで細かな地形の表示を実現している。そのため、カメラの位置が大きく変わる場合に多くの情報を再送信する。Santerre ら [18] は、メッシュシェーダを用いて地形の動的生成を実現しているが、地形情報は事前に用意しておく必要がある。

本研究では、生成する地形の領域や位置を変更したい場合、少ない情報の再送信によって、動的に地形の生成の変更が行えることを目的とした。GPU にいくつかの情報を送っておき、少ない情報の再送信により、送信した情報をもとに GPU 内でテッセレーションシェーダを使い、複数のパラメトリック曲面を生成して地形の生成を行う。パラメトリック曲面はいくつかの制御点により形状を定義しておき、いくつかのパラメータを与えて対応する位置を算出していく。算出した位置をつなぎ合わせることで生成する曲面形状をパラメトリック曲面と呼ぶ。また、テッセレーションシェーダは GPU 内でモデルの形状に対して修正をおこなったり、新たな頂点の生成が行えるシェーダである。そのため、パラメトリック曲面の生成に必要な頂点情報などを事前に送っておき、テッセレーションシェーダを使用して GPU 内で形状を新たに生成する事で、再送

信の情報量を減らすことができる。また、複数のパーリンノイズを合成して地形の高さ方向に加算する事で生成する地形のサイズに合わせた地形形状の表示も行った。振幅や周波数の違うパーリンノイズを複数合成して地形の高さ方向に使用することで、広い範囲の地形を表示した場合には大まかな地形の形状を表示し、狭い範囲の地形を表示する場合には、広い範囲の時には見えなかったような細かい地形の形状を表示が行えるようにした。GPU に送る情報は、各パラメトリック曲面の制御点情報、各曲面の領域情報、生成する地形の範囲の 3 つを GPU に送る。各パラメトリック曲面の制御点情報と各曲面の領域情報は始めに CPU からを GPU に一度だけ送信して保存しておく。生成地形に変更があるたびに生成する地形の範囲を再設定し、GPU に送る。そして、送信した情報をもとに GPU 内でテッセレーションシェーダを使い地形を生成する。そして、地形の高さ方向をもとにフラグメントシェーダで色付けを行った。

従来の手法では事前に地形を分割しておき、カメラ距離に合わせて分割した地形の情報を送るため、カメラの位置が大きく変化した場合、GPU に多くの情報を再送信する必要がある。また、事前に用意しておいた形状情報の段階までしか詳細な表示が行えず、より狭い範囲の地形を詳細に表示を行う場合、さらに地形の分割数が増え、それに対応する形状データを準備して、GPU に送る情報量が増加する。本手法ではテッセレーションシェーダとパラメトリック曲面、パーリンノイズを使用して GPU 内で地形を生成を行っている。そのため、GPU への再送信時に送る情報量は少なく、狭い領域を表示したい場合であっても送信する情報のパラメータが変化するだけで、送信する情報量は増えない。また、テッセレーションシェーダを使い GPU 内で生成をしているため、いくら小さい地形を生成した場合でも、同じ頂点数で曲面を生成することができるため一枚の曲面に関して、いくら狭い領域の地形を生成する場合でも、同じポリゴン数の地形を描画することができる。パラメトリック曲面とパーリンノイズを使用し、少ない情報の送信によって、地形の生成の実現を行った。

1.2　論文構成

本論文は全 4 章にて構成する。2 章では本手法について述べ、3 章では本手法を用いた実行結果について述べる。そして、4 章にてまとめを述べる。

第 2 章

提案手法

2.1　提案手法の方針

まず本手法の方針について述べる。本手法は以下のような流れで地形を生成していく。まず初めに広大な地形を表現するために複数枚の曲面を使い地形の大まかな形状を表現する。次に曲面で生成した形状に対してノイズを使い細かい形状を加えて、地形の最終的な形状を設定する。最後に生成した地形の高さ方向をもとにして地形に色をつけて生成するという流れになっている。

本手法では広大な地形を曲面を使って表現する。広大な地形を一枚の曲面で表現するのは難しいため、複数枚の曲面を生成して、つなぎ合わせることで地形を生成する。この曲面によって表現する地形の形状は地形の大まかな形状のものとなっており、この地形に対して細かい形状を加えることで表現したい地形を生成する。以下の図 2.1 は複数枚の曲面を接続し上から見たイメージを示した図である。

図 2.1　複数枚の曲面をつなぎ合わせたイメージ

次に大まかな地形に対して地形の細かい形状の加える。曲面で表現した地形にノイズを使用して細かい形状を加えることで、最終的な形状を設定する。そのため、曲面では地形のある程度の形を表現できるように設定し、細かい形状の設定に関してはノイズ使用し曲面で表現した地形に細かい形状加えて表現する。以下の図 2.2 は曲面のみで表現した地形とノイズを加えて表現した地形のイメージを表した図である。

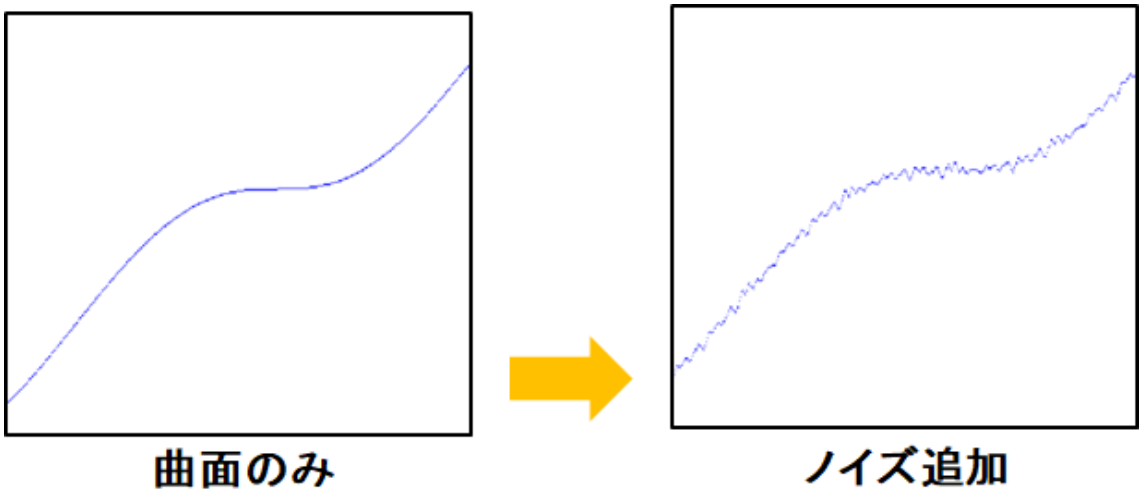

図 2.2　曲面のみの形とノイズを加えた形のイメージ

最後に生成した地形に対して色をつけて最終的な地形の完成となる。地形の色は設定した地形の高さに合わせて、どの色をどれくらい使うかを決めて色をつける。

以上の内容をパラメトリック曲面とテッセレーションシェーダ、パーリンノイズを用いて、少ない情報の再送信により、地形の生成を行う。処理の説明に関して、CPU 側の処理と GPU 側の処理に分かれて説明していく。CPU 側の処理は GPU に送信するための 3 つの情報を準備して送信しますという内容になっている。送信する処理は 3 つあり、各パラメトリック曲面の制御点、各曲面の領域情報、生成する地形の範囲情報の 3 つである。GPU 側の処理では送られてきた情報を使い GPU 内で曲面による生成する地形の形状を計算し、パーリンノイズを使用し細かい形状を加えて地形の形状を設定する。そして、最後に地形に色を付けて表示を行う。以下に CPU 側と GPU 側の処理をまとめたものを表しておく。

- CPU 内で 3 つの情報を準備して送信
  1. パラメトリック曲面の制御点
  2. 各曲面の領域情報
  3. 生成する地形の範囲情報

- GPU 内で送信された情報を使い地形生成処理
  1. GPU 内で曲面による地形形状の計算を行う
  2. パーリンノイズを使い細かい形状を地形に加算
  3. 生成した地形に色付け

始めに GPU に送る情報である、各パラメトリック曲面の制御点情報、各曲面の領域情報、生成する地形の範囲に関して説明をしていく。

2.2 パラメトリック曲面

パラメトリック曲面はいくつかの制御点を設定することにより定義される形状のことである。一組のパラメータを与えると対応する位置が算出でき、それらを繋ぎ合わせることで出来上がる曲面がパラメトリック曲面である代表的なものとして Coons[19] がインタラクティブに形状を扱う曲面表現式として提案した Coons 曲面や Bézier[20][21] が自動車の設計を行うために作成した Bézier 曲面などがある。このパラメトリック曲面を制御点を GPU に送り、送った制御点情報から GPU 内で実際の曲面の計算して形状を生成や変更を行うため、カメラ距離の変更時に再送信の情報量を減らすことができる。本研究では複数枚のパラメトリック曲面を生成し、繋ぎ合わせることで広大な地形の表現を行う。そのため、複数のパラメトリック曲面を繋ぎ合わせため、本研究では使用するパラメトリック曲面として Gregory 曲面 [22][23] を選択した。

2.3 Gregory 曲面

Gregory 曲面は、Gregory がおこなった一般 Coons 曲面の拡張を Bézier 曲面に用いた曲面である。Bézier 曲面は、パラメトリック曲面の一つであり、制御点と呼ぶ位置ベクトルのみで定義する曲面形状のことである。n×m 次の Bézier 曲面の曲面表現式は式 (2.1) のようになる。

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)B_j^m(v)\mathbf{P}_{ij} \tag{2.1}$$

$\mathbf{P}_{ij}$ は Bézier 曲面の制御点を表し、u と v は $0 \leqq u, v \leqq 1$ である。制御点は u 方向に $n+1$ 個、v 方向に $m+1$ 個、合計 $(n+1)(m+1)$ 個存在している。また、$B_i^n(u)$、$B_j^m(v)$ は Brnstein 基底関数であり、式 (2.2) は Bernstein 基底関数をあらわしたものである。

$$B_i^n(t) = \left(\begin{array}{c} n \\ i \end{array} \right) t^i(1-t)^{n-i} \tag{2.2}$$

ここで式 (2.3) は 2 項係数である.

$$\left(\begin{array}{c} n \\ i \end{array} \right) =_n C_i = \frac{n!}{i!(n-i)!} \tag{2.3}$$

Bézier 曲面はこの制御点を移動することで、形状を変更することが可能である。しかし、隣り合う Bézier 曲面同士を滑らかに接続するには複雑な計算が必要となってしまう。

そこで 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 個の制御点で表現し、制御点は $\mathbf{P}_{ijk}(i = 0 \ldots 3, j = 0 \ldots 3, k = 0, 1)$ にて表す。式 (2.4) は Gregory 曲面の曲面表現式を表したものである。

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

$B_i^n(u)$ と $B_j^m(v)$ は式 (2.2) の Bernstein 基底関数である。

また、曲面の制御点 $\mathbf{P}_{ijk}$ と $\mathbf{Q}_{ij}$ は曲面の制御点であり，次のような関係がある.

•i ≠1,2 または j ≠1,2 のとき

$$\mathbf{Q}_{ij} = \mathbf{P}_{ij0} \tag{2.5}$$

•$i = 1$,2 かつ $j = 1$,2 のとき

$$\begin{aligned}
\mathbf{Q}_{11}(u,v) &= \frac{u\mathbf{P}_{110} + v\mathbf{P}_{111}}{u+v} \\
\mathbf{Q}_{12}(u,v) &= \frac{u\mathbf{P}_{120} + (1-v)\mathbf{P}_{121}}{u+(1-v)} \\
\mathbf{Q}_{21}(u,v) &= \frac{(1-u)\mathbf{P}_{210} + v\mathbf{P}_{211}}{(1-u)+v} \\
\mathbf{Q}_{22}(u,v) &= \frac{(1-u)\mathbf{P}_{220} + (1-v)\mathbf{P}_{221}}{(1-u)+(1-v)}
\end{aligned} \tag{2.6}$$

また、共通の境界線をもつ曲面同士の接続について述べる。境界線に隣接している制御点間のベクトルを $\mathbf{A}_i(i = 0 \ldots 3), \mathbf{B}_i(i = 0 \ldots 3), \mathbf{C}_i(i = 0, 1, 2)$ とすると、

$$\mathbf{A}_1 = \frac{2\mathbf{A}_0 + \mathbf{A}_3}{3}, \mathbf{A}_2 = \frac{\mathbf{A}_0 + 2\mathbf{A}_3}{3} \tag{2.7}$$

と仮定した場合、式 (2.8) によって 3 つの曲面を滑らかに繋げるような $\mathbf{B}_1, \mathbf{B}_2$ を求めることができる。

$$\begin{aligned}
\mathbf{B}_1 &= \frac{(k_1 - k_0)\mathbf{A}_0 + 3k_0\mathbf{A}_1 + 2h_0\mathbf{C}_1 + h_1\mathbf{C}_0}{3}, \\
\mathbf{B}_2 &= \frac{3k_1\mathbf{A}_2 - (k_1 - k_0)\mathbf{A}_3 + h_0 C_2 + 2h_1\mathbf{C}_1}{3}
\end{aligned} \tag{2.8}$$

ただし、k_0, k_1, k_2, k_3 は

$$\begin{aligned}
\mathbf{B}_0 &= k_0\mathbf{A}_0 + h_0\mathbf{C}_0, \\
\mathbf{B}_3 &= k_1\mathbf{A}_3 + h_1\mathbf{C}_2
\end{aligned} \tag{2.9}$$

を満たす実数である。

2.4 各曲面の領域情報

各曲面の領域情報は各 Gregory 曲面が全ての曲面をつなぎ合わせた全体のどこから、どこまでの部分にあたるのかという情報であり、各 Gregory 曲面ごとにこれを設定してく。全ての曲面をつなぎ合わせた全体サイズを 0∼1 としたときに、各 Gregory 曲面の領域を 0∼1 の間の値をで設定していく。例えば、横に 4 枚、縦に 4 枚の合計 16 枚の曲面で全体の地形が表されており、この 16 枚を合わせて一枚の曲面として見る。この広大な曲面の UV 座標系で表すと、左下が U が 0 で V が 0、右上が U が 1 で V が 1 として、各曲面の最小値 a_{ij}、b_{ij} と最大値 e_{ij}、f_{ij} の設定を行う。横に n 個、縦に m 個曲面が並んでいる場合、各の曲面の最小値と最大値の設定 i と j は $(i = 0 \ldots n-1, j = 0 \ldots m-1)$ となる。左から 2 番目、下から 1 番目の曲面の設定を行う場合、最小値は a_{ij} は 0.25、b_{ij} は 0 となり最大値は e_{ij} は 0.5、f_{ij} は 0.25 となる。また、最小値と最大値の各値は 0∼1 の範囲内の定数が入る。そして、設定した各 Gregory 曲面の制御点と各 Gregory 曲面に対応した最小値 a_{ij}、b_{ij} と最大値 e_{ij}、f_{ij} は一度だけ GPU に送信し、送った情報を繰り返し使っていく。以下の図 2.3 は、最小値と最大値を設定している曲面を表した図である。左から 2 番目、下から 1 番目の青い枠線が設定を行っている曲面であり、各曲面が地形全体のどこからどこにあたるのかを設定しておく。

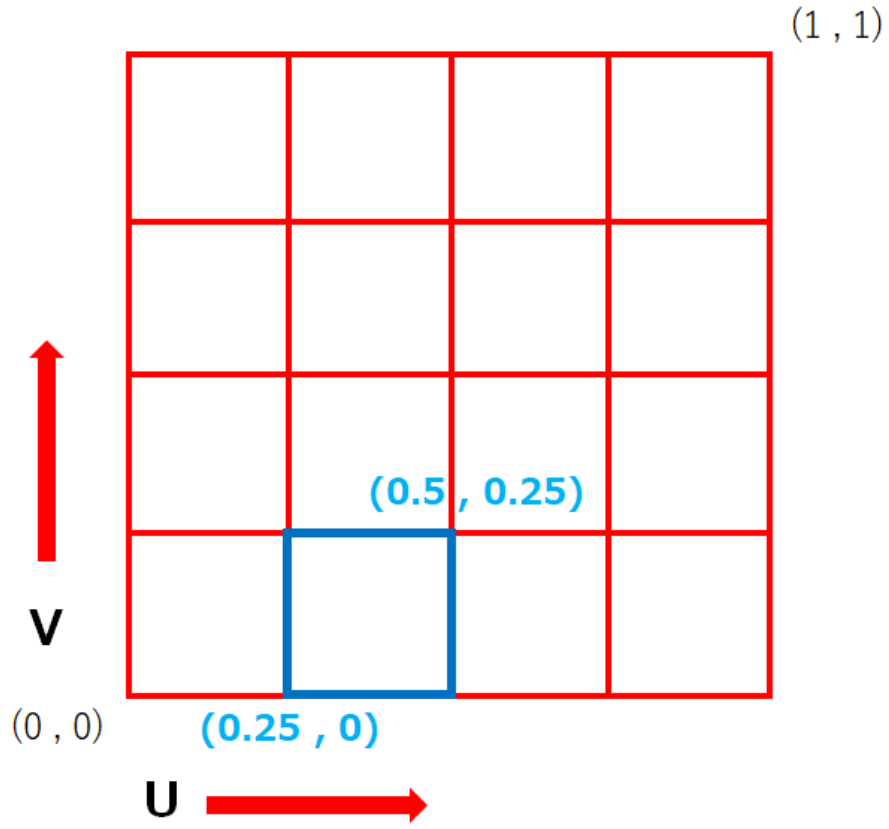

図 2.3　各の曲面の最小値と最大値の設定

2.5　生成する地形の範囲情報

生成する地形の範囲の情報は複数の曲面を一つの地形として見た場合に全体から、どこの部分の表示を行うのかを決める情報である。GPU に適宜この情報を送信し、この情報をもとに最終的に生成する地形の形状が決定する。全部の曲面を繋ぎ合わせた物を一つの地形として見て、各曲面の領域情報の設定のように最小値と最大値を設定する。例えば、地形全体の約 1/4 ほどの地形を表示したい場合、UV 座標系で最小値を c、d と最大値を g、h として、c は 0.125、e は 0.125 となり、g は 0.625、h は 0.625 と設定する。こちらの最小値と最大値の各値は 0~1 の範囲内の定数が入る。この最小値を c、d と最大値 g、h は値を変更し、GPU に再送信することで複数の曲面全体から、任意の位置と大きさの地形を生成することができる。ここで求めた値によって全体の

地形のどこを表示するのかということが決定する。以下の図 2.4 は生成する地形の範囲情報を設定例の様子であり、左下の緑の領域が設定した生成する地形の範囲となる。

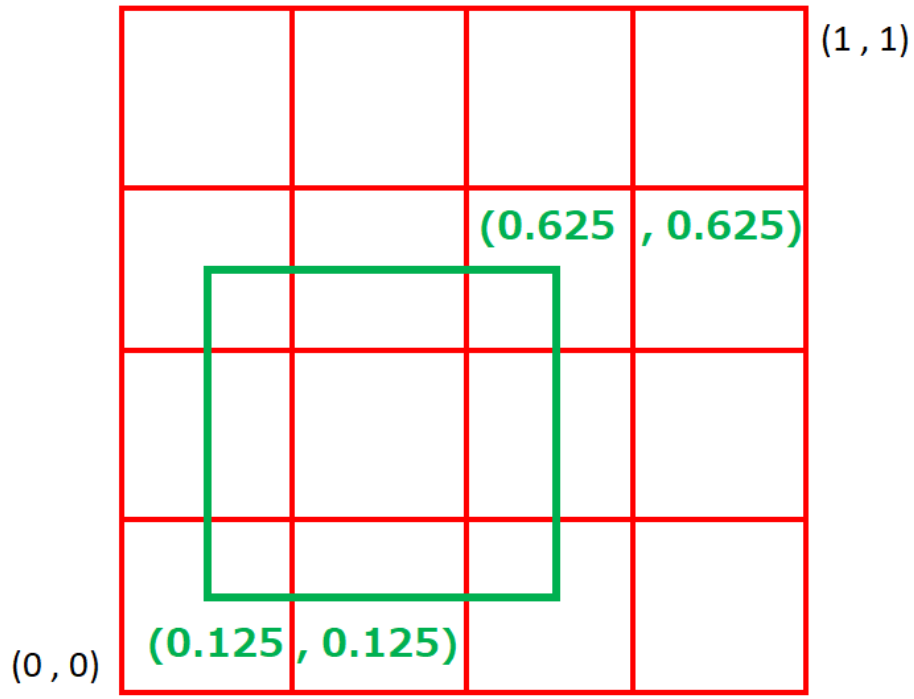

図 2.4　生成する地形の範囲の設定

この c、d、g、h 値を適切に設定することで、生成したい領域のみの地形が生成できる。そのため、多くの情報を再送信しなくても、描画したい領域に合わせた地形を表示することが可能であり、カメラとの距離に対応して設定すれば、カメラ距離に応じた地形を生成することができる。また、生成する地形の範囲を適切に設定することで、いくら狭い領域の地形でも生成することが可能である。

2.6　カメラ情報をもとにした生成する地形の範囲情報の算出

カメラ情報を使い GPU に送る 3 つ目の情報である生成する地形の範囲情報 c、d、g、h を算出していく。大まかな処理の流れとしては、算出方法は地形全体から一定の間隔で 3 次元空間中の座標を取得し、取得した座標がカメラの描画範囲の内か外かを調べて、描画範囲内の最小値と最

大値を取得する。そして最終的に取得した描画範囲内の最小値と最大値の位置の地形がすべて映るように最小値と最大値の数値を少し広く設定し、地形の範囲情報として設定する。以上が大まかな処理の流れとなっている。それではこれからどのように求めていくのかを述べていく。

まず、複数の曲面をつなぎ合わせて一つの地形として見た場合に全体サイズを 0~1 として全体を一定間隔に区切っていく。この時、複数枚の曲面をつなぎ合わせた地形全体を上から見た平面の 2 次元座標として扱う。そして、区切った位置の値を $\mathbf{F}_{ij}$ とする。以下の図 2.5 は全体を一定間隔で区切った様子を表した図である。丸い点がそれぞれ区切った位置となっており、オレンジ色の位置を選択する場合、値が $\mathbf{F}_{50} = (0.625, 0)$ となる。

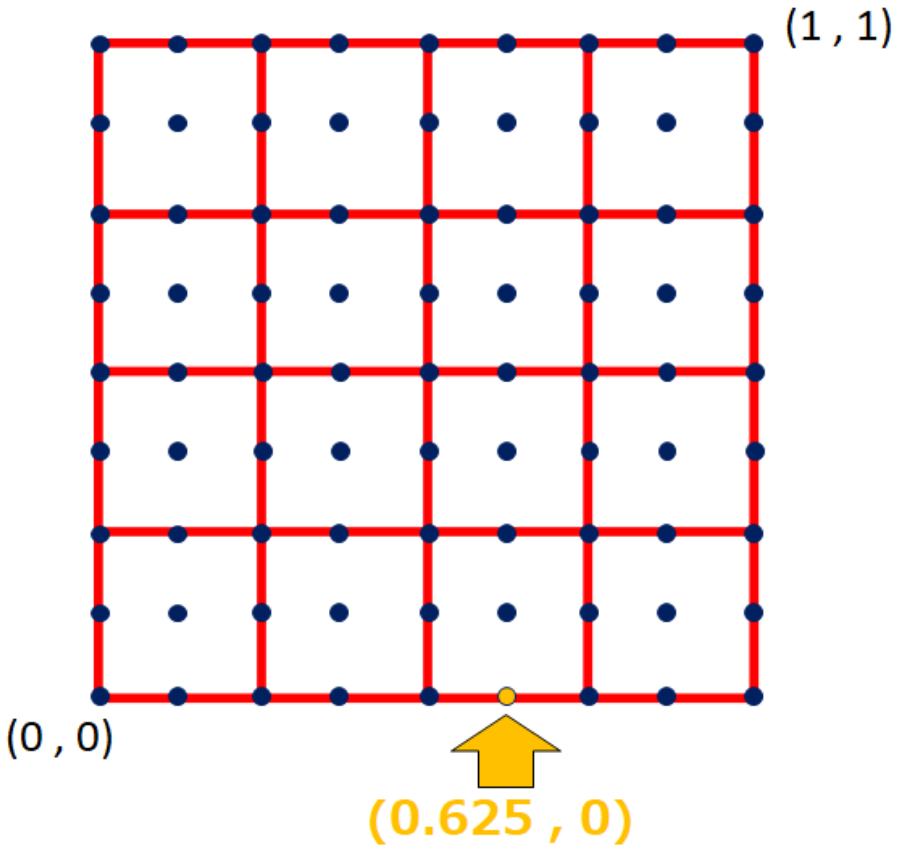

図 2.5　地形全体を一定間隔で区切った例

次に選択した位置の値 $\mathbf{F}_{ij}$ をもとに曲面計算式を使用して実際の 3 次元空間上の座標を算出す

る。まず初めに式 (2.4) を使用するための u, v を算出する。選択した位置の値 $\mathbf{F}_{ij}$ がどの曲面上にあるのか確認し、$\mathbf{F}_{ij}$ の位置の曲面の領域情報である最小値 a_{ij}、b_{ij} と最大値 e_{ij}、f_{ij} を取得する。そして、$\mathbf{F}_{ij}$ の値を 1 枚の曲面の u, v に変換する。式 (2.10) は u, v を算出するための式であり、ここで求めた u, v を使い式 (2.4) で 3 次元空間中の座標を取得する。

$$
\begin{aligned}
u &= \frac{c - a_{ij}}{e_{ij} - a_{ij}} \\
v &= \frac{d - b_{ij}}{f_{ij} - b_{ij}}
\end{aligned}
\tag{2.10}
$$

式 (2.10) を使って求めた 3 次元空間中の座標がカメラの描画内にあるかどうか判断する。まず、式 (2.10) を使って求めた 3 次元空間中の座標をカメラの描画内の計算のためにカメラ座標を原点とした座標系に変更する。カメラ座標を原点とした座標系に変更したものを $\mathbf{D}$ としたとき、スクリーン座標系ではどこに当たるのかを求める。カメラから描画が行われる最も近い距離を ϕ、最も遠い距離を χ、最も近い距離での横幅を ψ、縦幅を ω として、各値を取得して計算を行う。式 (2.11) はスクリーン座標を算出するための式である。カメラを原点とした $\mathbf{D}$ を使用して、スクリーン空間正規化座標 $\mathbf{E}$ を求めている。

$$
\mathbf{E} = \begin{pmatrix}
\frac{2\phi}{\psi} & 0 & 0 & 0 \\
0 & \frac{2\phi}{\omega} & 0 & 0 \\
0 & 0 & -\frac{\chi+\phi}{\chi-\phi} & -\frac{2\phi\chi}{\chi-\phi} \\
0 & 0 & -1 & 0
\end{pmatrix} \mathbf{D} \tag{2.11}
$$

式 (2.11) で求めた $\mathbf{E}$ を使用して、式 (2.12) を使いカメラの描画内にあるか判断する。式 (2.12) で求めた値 $\mathbf{G}$ の x, y, z の成分がすべて $-1 \sim 1$ の間にあれば、カメラの描画内に存在する。

$$
\mathbf{G} = \left(\frac{E_x}{E_w}, \frac{E_y}{E_w}, \frac{E_z}{E_w} \right) \tag{2.12}
$$

カメラの描画内にあるベクトル $\mathbf{F}_{ij}$ だけを取得していき、取得した $\mathbf{F}_{ij}$ がすべてカメラの描画範囲内になるように最小値 c、d と最大値 g、h を地形の範囲情報として設定する。以下の図 2.6 はカメラの描画範囲内の点を取得した後に地形の範囲情報である、最小値 c、d と最大値 g、h を設定するイメージ図である。図 2.6 の上側の緑の枠が描画範囲であり、色が濃い丸が描画範囲内の座標、色が薄い丸が描画範囲外の座標である。下側のオレンジ色の点がカメラの描画範囲内の座標がすべて入るように設定した最小値 c、d と最大値 g、h の様子である。

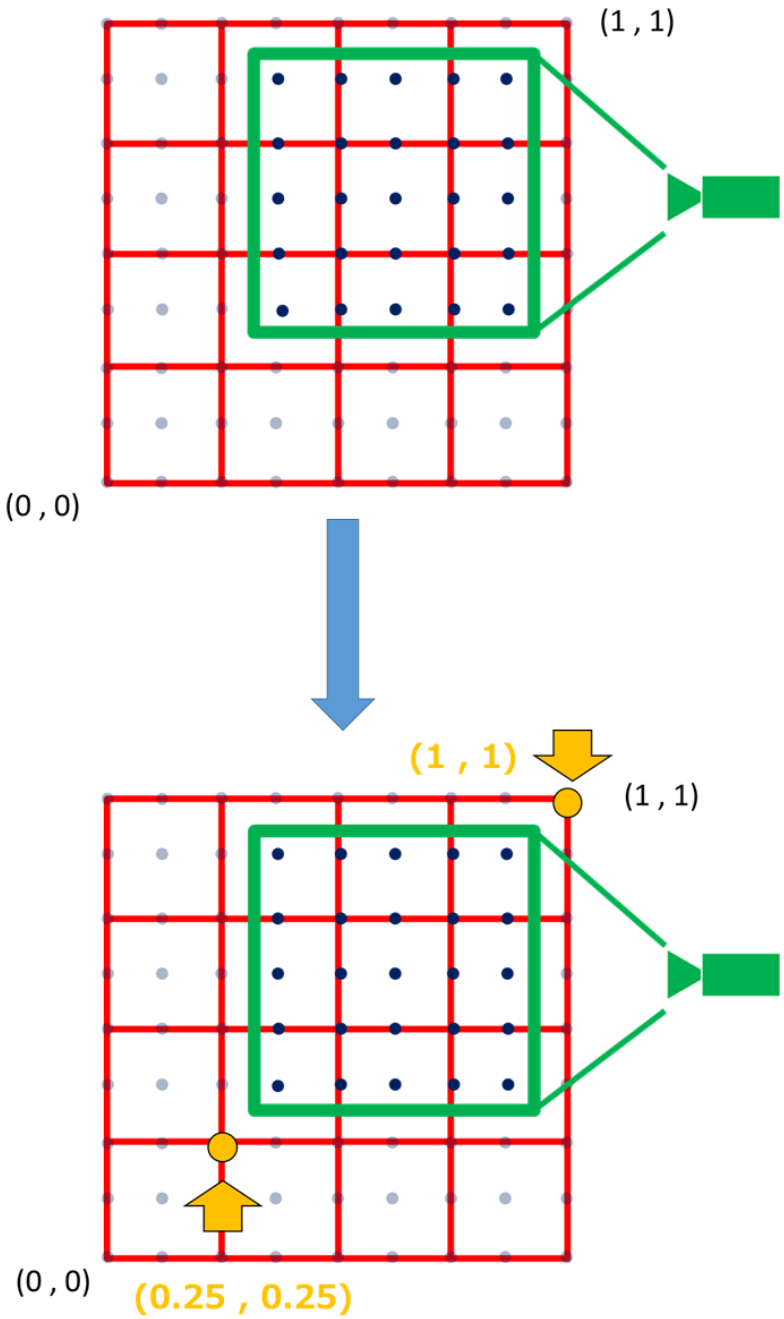

図 2.6　カメラの描画範囲にある点をもとに最小値と最大値を決めた例

また、全体を一定間隔に区切った座標のうち、ある一定数の座標がカメラの外だった場合 0~1 の間で区切っていたものを、最小値を c、d と最大値を g、h の間で区切っていく。以下の図 2.7 はカメラの描画範囲にあわせて区切っていた範囲を変えた例である。

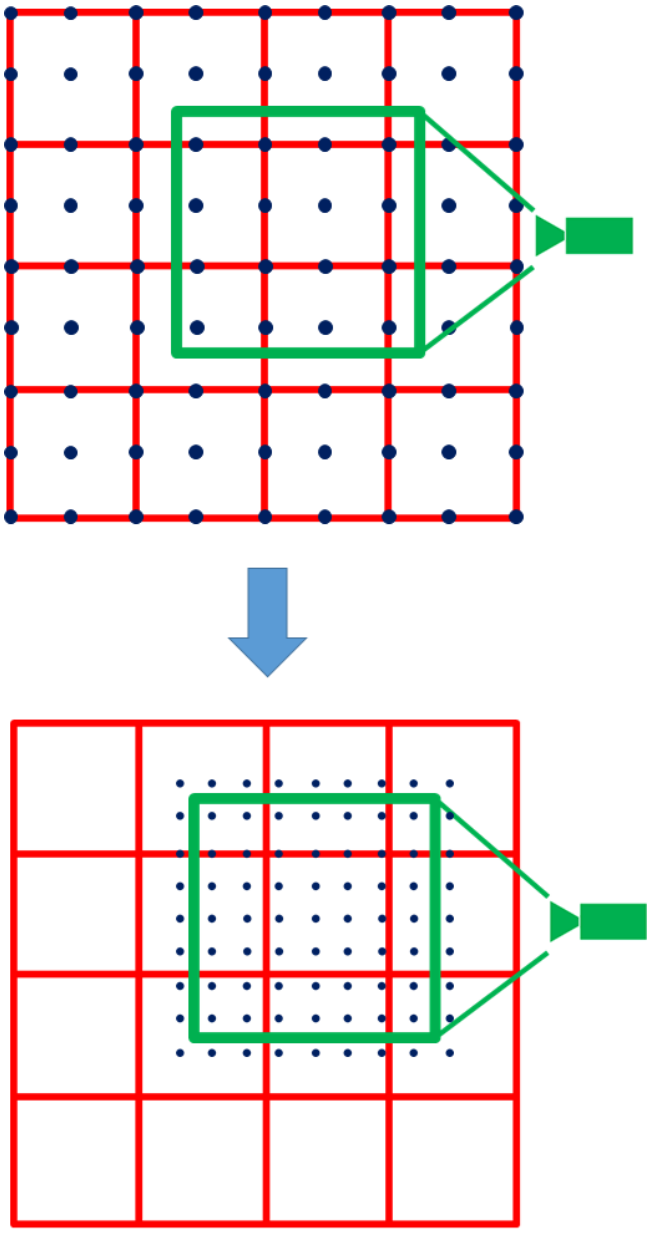

図 2.7　カメラの描画範囲にある点をもとに最小値と最大値を決めた例

2.7　CPU 側での地形の生成の準備

始めに CPU 内で GPU に送る情報を準備していく。まず、各 Gregory 曲面の制御点を設定を行う。2.1 節でも述べたが、広大な地形を一枚のパラメトリック曲面で表現するのは難しいため、複数枚の曲面を生成し、繋ぎ合わせることで広大な地形の表現を行う。各 Gregory 曲面が地形

の一部を表現するように設定を行い、全てを繋ぎ合わせた時に地形の大まかな形状を表す。この Gregory 曲面で表現した形状に対し、GPU 内で細かい地形の表現を追加し、最終的な形状を完成させる。

次に各 Gregory 曲面に対して、各曲面の領域情報の設定をしていく。各 Gregory 曲面が横と縦にいくつ並んでいるのかをもとに、各 Gregory 曲面が全体のどこからどこの領域に当たるか、各曲面の最小値 a_{ij}、b_{ij} と最大値 e_{ij}、f_{ij} の設定を行う。そして、設定した制御点情報と各曲面の領域情報を GPU に一度だけ送信する。

最後に生成する地形の範囲情報の最小値を c、d と最大値 g、h を設定し、GPU に情報を送る。生成する地形に変更をかけたい場合、この生成する地形の範囲情報に対し再設定を行い、GPU に情報を再送信する必要がある。

2.8 GPU 側での地形の生成

CPU から GPU に送った情報をもとにテッセレーションシェーダを使い地形の形状を生成を行う。テッセレーションシェーダ [24] [25] は、GPU 内でモデルの形状に対して修正を加えたり、新たな頂点の生成や頂点の削除を行うことができるシェーダである。これを使うことにより、GPU 内でリアルタイムにモデル形状の変更を行うことができる。テッセレーションシェーダはテッセレーション制御シェーダとテッセレーション評価シェーダ 2 つのシェーダとテッセレーションプリミティブジェネレータによって構成されている。テッセレーション制御シェーダはどのような形状を生成するのかということを定義する役割を持ち、ここで設定した情報をテッセレーションプリミティブジェネレータに送ることで、新たな形状の生成が行われます。GPU 上にある形状データを入力情報として各頂点ごとに一度実行が行われ、入力情報をもとに新たに生成する形状の情報を定義して、テッセレーションプリミティブジェネレータに情報の送信を行う。次に、テッセレーションプリミティブジェネレータが受け取った情報をもとに、0 から 1 のパラメータ空間内

に新たな頂点を生成していく。ここで生成した頂点情報をテッセレーション評価シェーダに送信し、最終的な形状の位置が決定する。テッセレーション評価シェーダはテッセレーションプリミティブジェネレータが送った情報と GPU 上にある形状データを照らし合わせて、最終的な各頂点の位置の決定を行う。テッセレーションプリミティブジェネレータが送った各頂点ごとにテッセレーション評価シェーダが一度実行され、形状データの位置情報を使って、パラメータ空間から最終的な頂点の位置を計算して決定する。図 2.8 は、テッセレーションを用いて新たな形状を生成した例を示したものである。

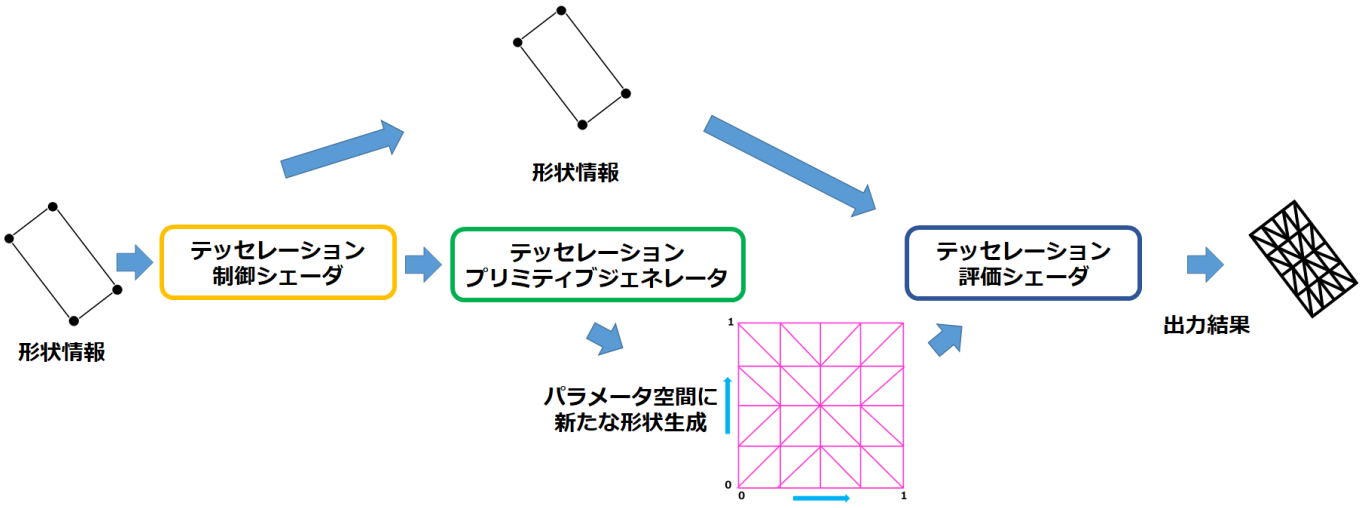

図 2.8　テッセレーションシェーダによる新たな形状生成例

このシェーダを使い地形を生成するため、制御点情報と各曲面の領域情報を一度のみ送信して GPU 内に保存し、生成する地形の範囲情報を再送信することで、任意の形状の地形を生成できる。大きい地形を生成した場合、小さい領域の地形を生成した場合でも同じ頂点数で地形の生成が可能である。そのため、カメラ距離に合わせて生成する地形の範囲情報を設定すれば、表示されている領域は小さくなっているが画面内のモデルの使用している頂点数が変わらない地形生成が可能である。図 2.9 は、一枚の曲面に関して、小さい領域を生成した場合に使う頂点の例を示したものである。以下の図のように小さい領域の地形を生成した場合でも、同じ頂点数での表示

が可能である。

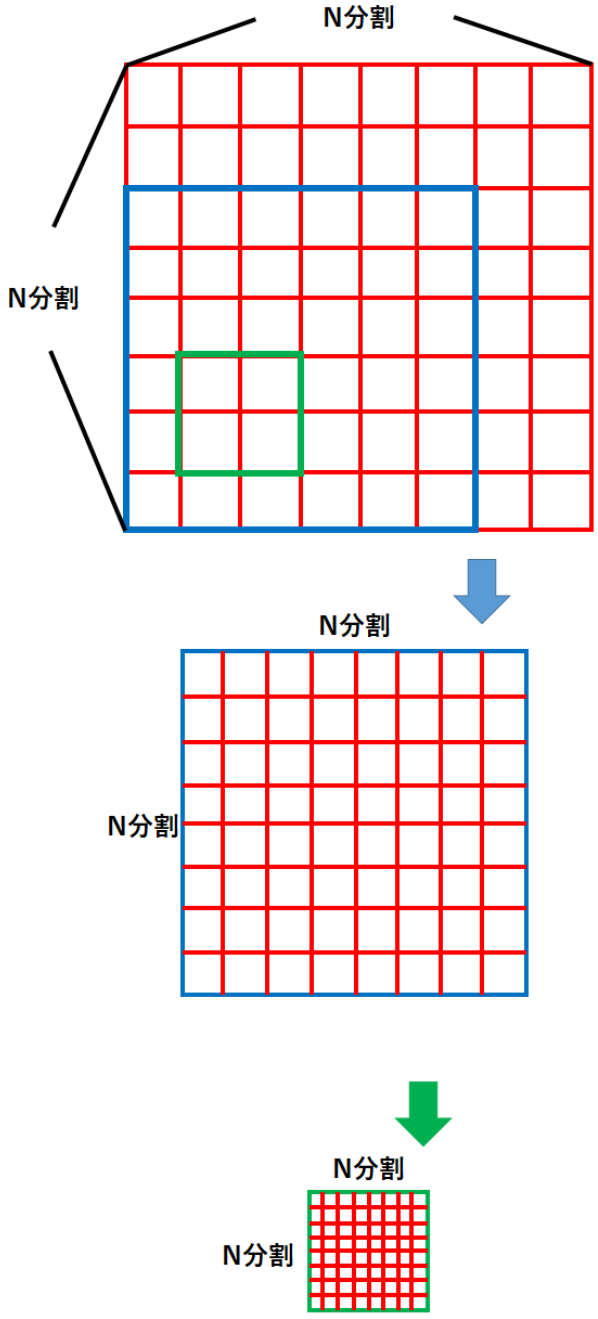

図 2.9　送信した情報をもとに曲面を生成した場合の頂点の例

このテッセレーションシェーダを使い、任意の領域の地形生成を行う。まず、GPU に送信した情報を使い、各 Gregory 曲面に対して式 (2.4) で使用する u,v を求める。u,v を求める式を表し

たものが式 (2.13) であり、u と v に入る値は 0~1 を超えなようにする。n、m はテッセレーションシェーダ内で新たに作られた頂点座標であり、gl_TessCoord で取得した x 座標が o、y 座標が p となっている。

$$
\begin{aligned}
u &= \frac{c - a_{ij}}{e_{ij} - a_{ij}} + o\frac{1 - g - e_{ij}}{a_{ij} - e_{ij}} \\
v &= \frac{d - b_{ij}}{f_{ij} - b_{ij}} + p\frac{1 - h - f_{ij}}{b_{ij} - f_{ij}}
\end{aligned}
\tag{2.13}
$$

式 (2.13) で求めた u と v を式 (2.4) で使い GPU に送った生成範囲の地形生成を行う。

最後に生成した地形に対して、細かい形状を加えていく。曲面のみでは滑らかな形状表現になってしまうため、地形の形状を表現するためにパーリンノイズ [26][27] を使用し、細かい形状を加えることで地形を表現する。パーリンノイズは KenPerlin が作成したノイズ生成のアルゴリズムであり、フラクタルな性質を持つ自然物の形状生成などに使用されているアルゴリズムである。安達ら [28] は、パーリンノイズを用いて埃の高速描画を実現した。また、間淵ら [29] は、炎のレンダリングの際、テクスチャの生成にパーリンノイズを用いている。本研究では、二次元のパーリンノイズの値を地形の高さ方向に追加し、地形の細かな形状を表現する。

パーリンノイズには各曲面の領域情報と各頂点の位置にあたる数値を使用する。式 (2.14) はパーリンノイズに使用する値を計算したものである。

$$
\begin{aligned}
q &= a_{ij} + u(e_{ij} - a_{ij}) \\
r &= b_{ij} + v(f_{ij} - b_{ij})
\end{aligned}
\tag{2.14}
$$

また、振幅や周波数の違うパーリンノイズを複数用意しておき、合成してノイズの値を出す。各パーリンノイズは地形全体が表示してある場合、狭い範囲の地形が表示してある場合、さらに狭

い範囲の地形が表示してある場合など、生成範囲に合わせて形状が表示されるように設定を行いそれらを合成することで、生成した地形に応じた形状が表示が行われる。式 (2.15) は 2 つのパーリンノイズを組み合わせた式である。

$$p = K(q, r) + \frac{K(\alpha q, \alpha r)}{\beta} \tag{2.15}$$

α はパーリンノイズの周波数を表しており、β は振幅を表している。α の値が大きいほど細かなパーリンノイズを生成でき、β が大きいほど生成した地形の範囲が狭い場合にそのパーリンノイズが表示される。

最後に式 (2.15) で求めた値を式 (2.4) で求めた Gregory 曲面の高さ方向に加算し、地形の形状を決定する。式 (2.16) は最終的な形状の高さ方向の計算式である。

$$S'_z = S_z + p \tag{2.16}$$

地形の色はフラグメントシェーダ内で色を設定する。テッセレーションシェーダ内で色をつけるためのパーリンノイズを用意して、頂点位置を使い数値を算出する。頂点の高さ情報 S'_z とパーリンノイズの値 l をフラグメントシェーダに渡して、各頂点の高さに応じて色を設定する。地形の基準となる色を RGB で設定したものを N とおき、高さに応じて塗る別の色の RGB が M とする。地形の高さに応じて N と M をどの位の割合でそれぞれ塗るのかを決めるのが値が s である。s は 0~1 の数値であり、式 (2.17) は s を求める式である。γ を調整することで高さに応じた割合を設定でき、s を使い式 (2.18) で最終的な色を決定する。

$$s = l + \frac{S'_z}{\gamma} \tag{2.17}$$

$$L = Ms + N(1 - s) \tag{2.18}$$

第 3 章

出力結果と検証

3.1 開発環境

前章で述べた手法を実装し、検証を行った。開発言語は C++ を用い、OS は Windows10 を使用した。また、実装は OpenGL をベースとする FK ライブラリ [30] 上で行った。開発および検証に用いた PC のスペックを表 3.1 に示す。

表 3.1 実行環境

OS	windows 10
CPU	Intel(R)Core(TM)i7-10750H CPU @ 2.59GHz
メモリ	16 GB
GPU	GeForce GTX 1660 Ti

3.2 出力結果

今回は 25 枚の曲面を用意し、テッセレーションシェーダ内で 4 つのパーリンノイズを準備して、本手法を使い地形生成を行った．GPU に送信する生成する地形の範囲情報の最小値 c を 0、d を 0 と設定しておき、最大値である g と h を $g = 1, h = 1$ と設定した後に、g と h を 0.5 ずつ減少させて生成した地形の実行結果を示す。以下の図 3.1 から図 3.19 は最大値を 0.5 ずつ減少させて生成した地形の実行結果を示した図である。GPU に送信する情報量は増えておらず、与えるパラメータの変更のみで生成する地形が変わっており、パラメータの変化によって、広い範囲の地形から狭い範囲の地形の生成になっている。

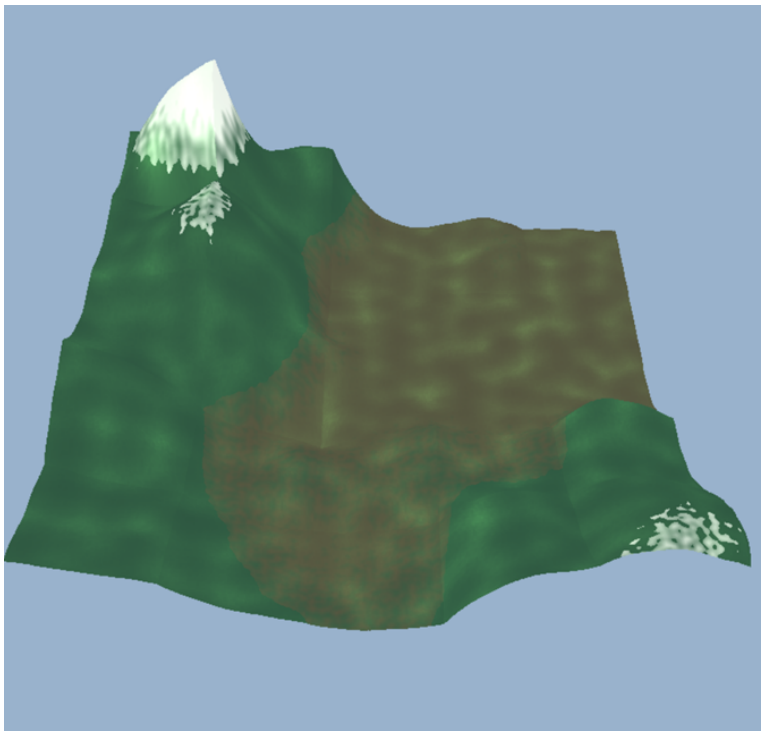

図 3.1　g と h が 1.0 の地形

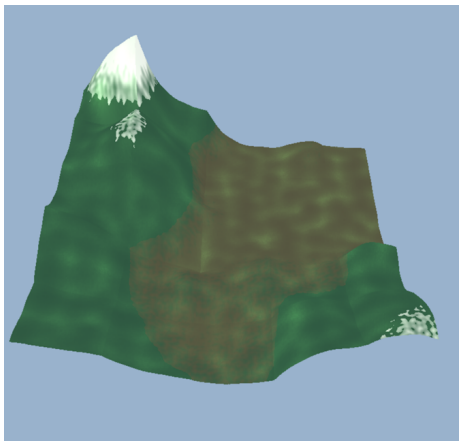

図 3.2　g と h が 0.95 の地形

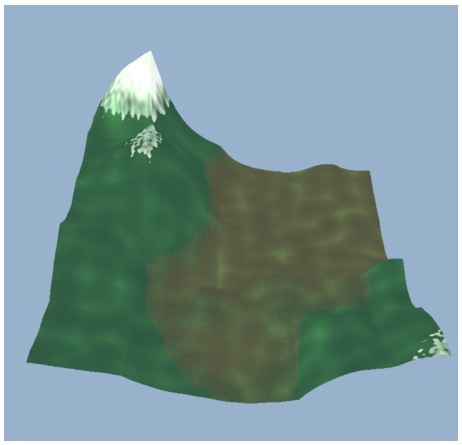

図 3.3　g と h が 0.9 の地形

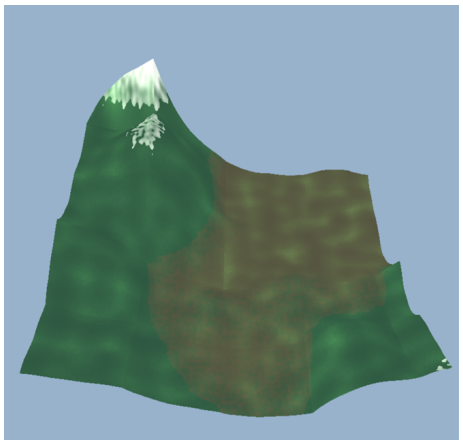

図 3.4　g と h が 0.85 の地形

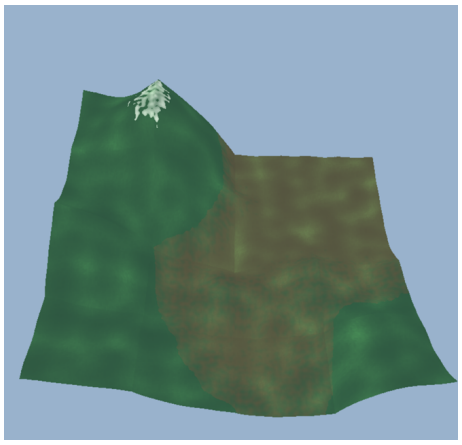

図 3.5　g と h が 0.8 の地形

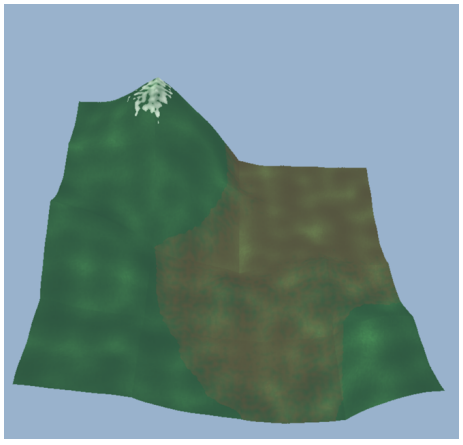

図 3.6　g と h が 0.75 の地形

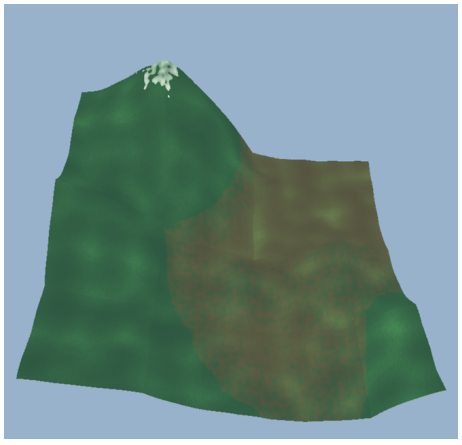

図 3.7　g と h が 0.7 の地形

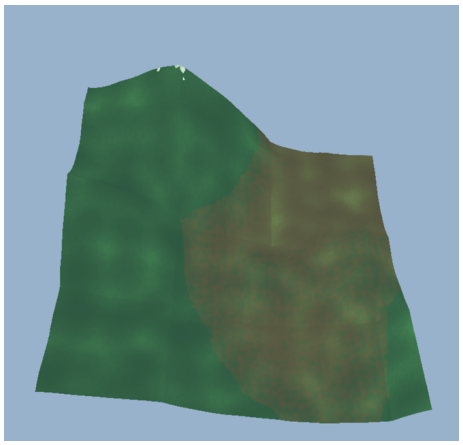

図 3.8　g と h が 0.65 の地形

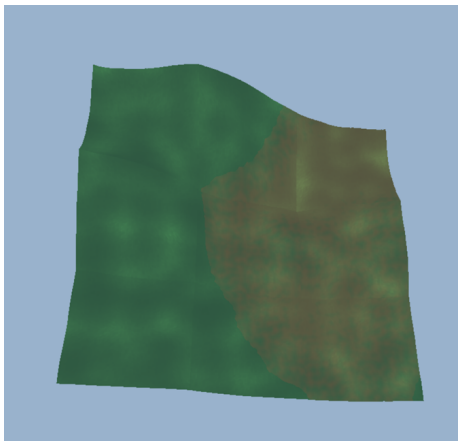

図 3.9　g と h が 0.6 の地形

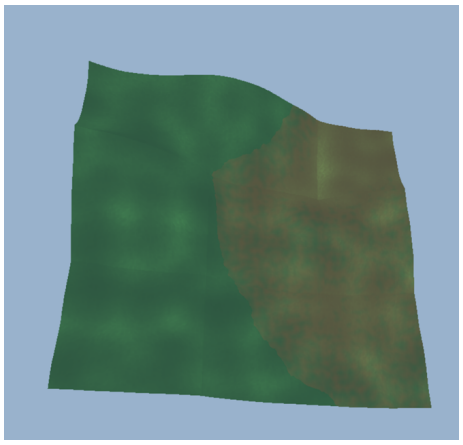

図 3.10　g と h が 0.55 の地形

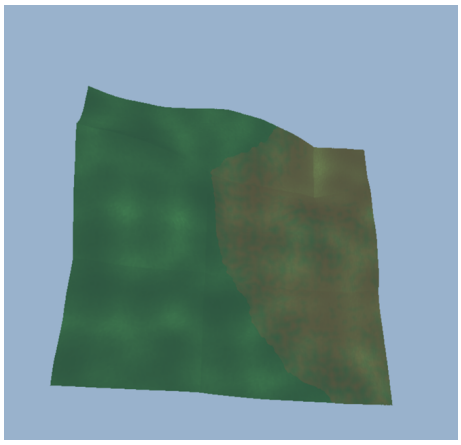

図 3.11　g と h が 0.5 の地形

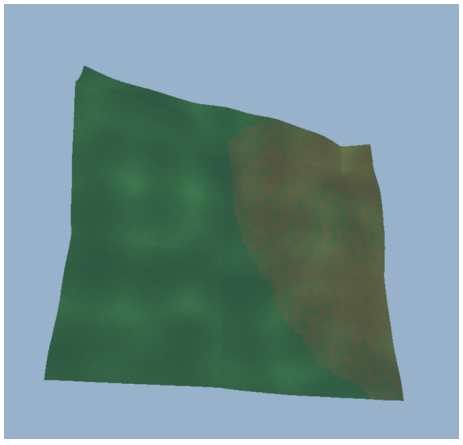

図 3.12　g と h が 0.45 の地形

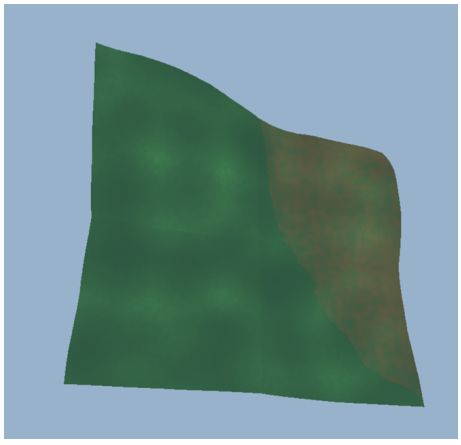

図 3.13　g と h が 0.4 の地形

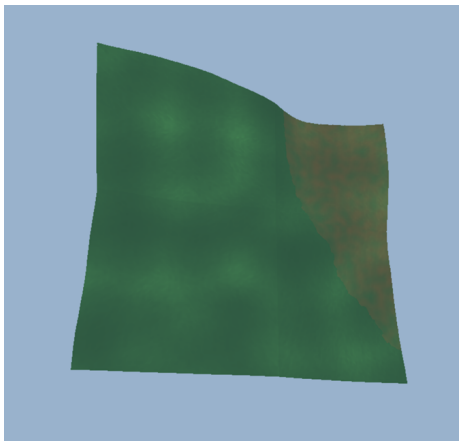

図 3.14　g と h が 0.35 の地形

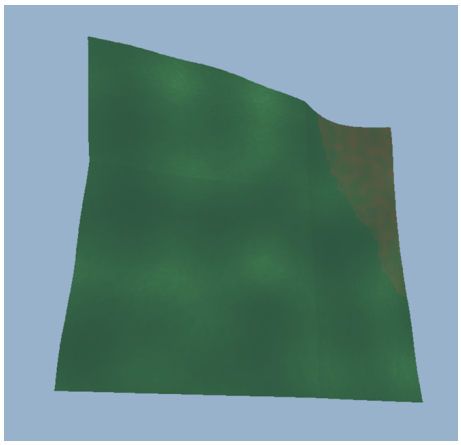

図 3.15　g と h が 0.3 の地形

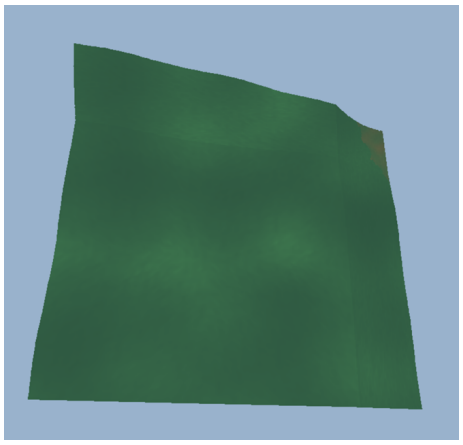

図 3.16　g と h が 0.25 の地形

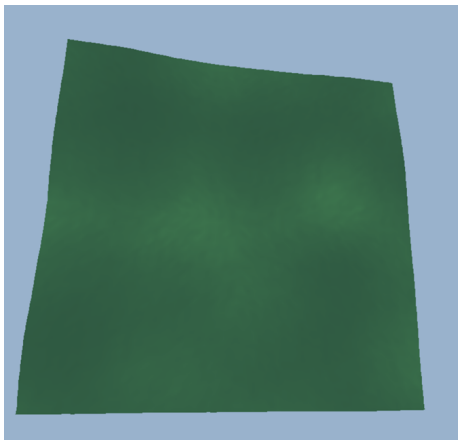

図 3.17　g と h が 0.2 の地形

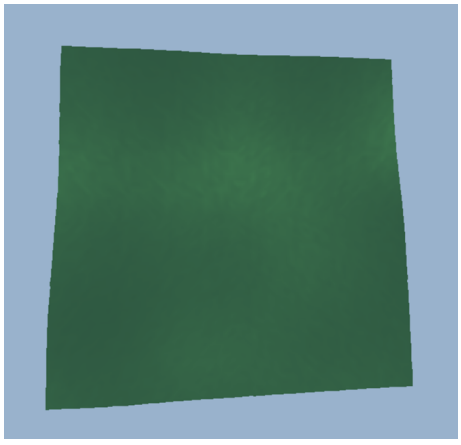

図 3.18　g と h が 0.15 の地形

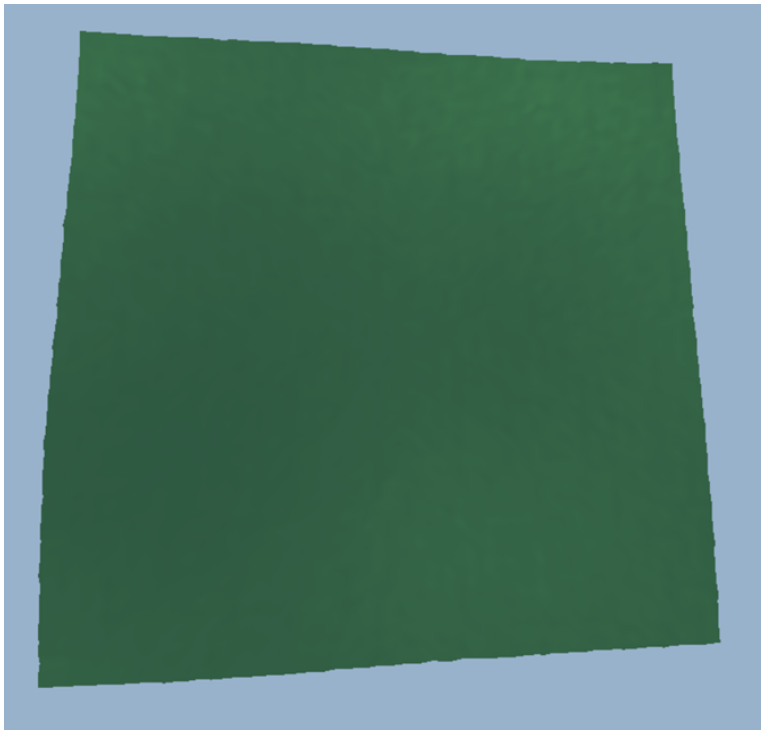

図 3.19　g と h が 0.1 の地形

以下の図 3.20 から図 3.23 は大きな地形生成した場合から小さな地形を生成した場合に頂点数がどうなっているのかを示した図である。右が実際に生成された地形であり、左がワイヤーフレームを表示したものである。最小値の c を 0、d を 0 と設定し、最大値である g と h の数値を変更した。図 3.20 は g と h が $g = 1.0, h = 1.0$ の生成を行った場合の地形であり、複数の曲面で広い範囲の地形が表示してある。図 3.21 は g と h が $g = 0.5, h = 0.5$ の生成を行った場合の地形であり、図 3.20 の半分のサイズの地形が表示がされている。図 3.22 は g と h が $g = 0.2, h = 0.2$ の生成を行った場合であり、一枚の曲面によって地形が表示してある。図 3.23 は g と h が $g = 0.05, h = 0.05$ の生成を行った場合であり、パーリンノイズにより細かい形状が浮かび上がってきている。図 3.24 は g と h が $g = 0.05, h = 0.05$ の生成を行った場合であり、さらに狭い領域が表示してある。いくら小さい範囲の生成を行った場合でも同じポリゴン数での生成が行われている。また、複数のパーリンノイズにより、それぞれのパーリンノイズによる形状が生成する地形の範囲が狭くなるにつれて見えるようになっている。

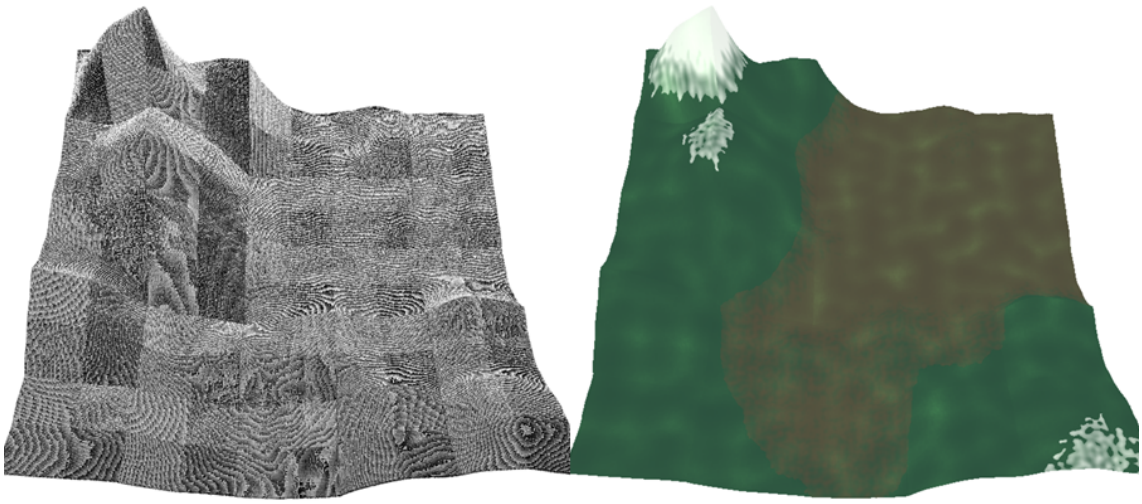

図 3.20　g と h が 1.0 の地形とワイヤーフレームの表示

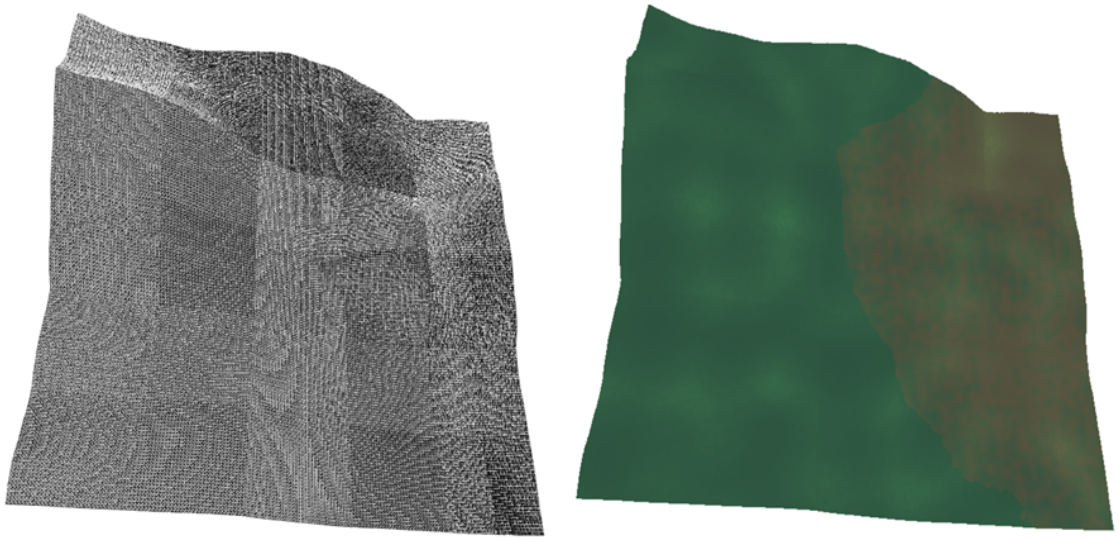

図 3.21　g と h が 0.5 の地形とワイヤーフレームの表示

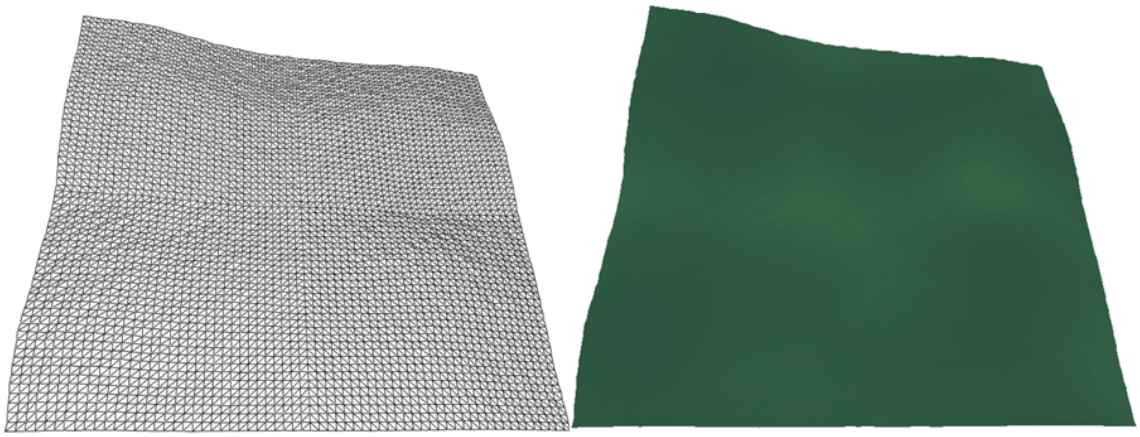

図 3.22　g と h が 0.2 の地形とワイヤーフレームの表示

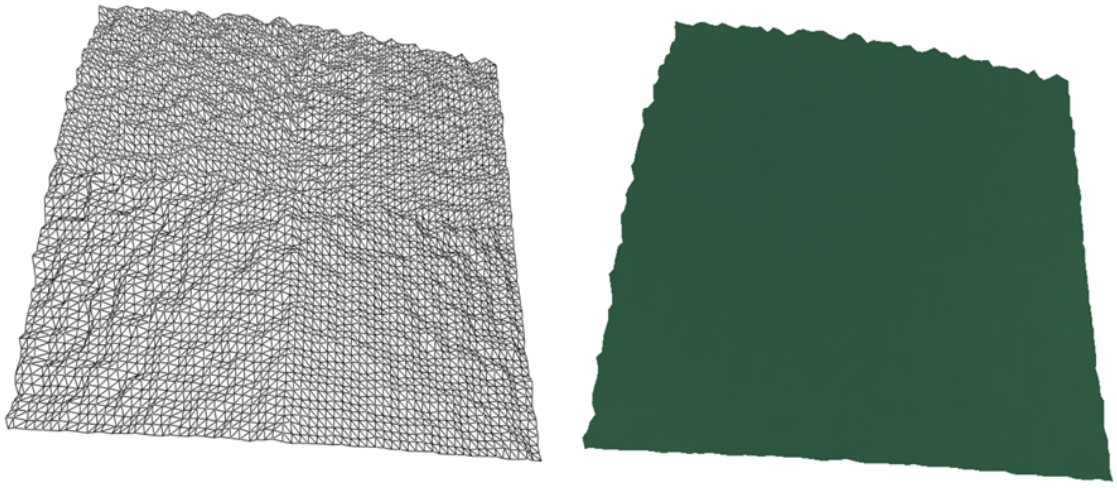

図 3.23　g と h が 0.05 の地形とワイヤーフレームの表示

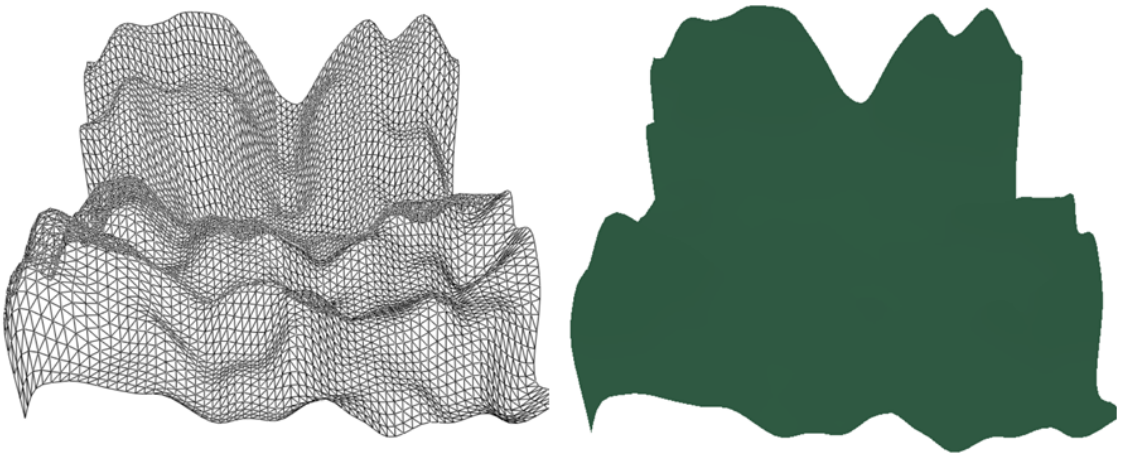

図 3.24　g と h が 0.005 の地形とワイヤーフレームの表示

3.3　本手法と従来手法の比較

一枚の曲面を制作して一度 GPU に送信を行い、形状に変更を加えて変更があった箇所の形状の再送信を行う場合の速度検証をおこなった。本手法を用いて曲面を生成し形状変化による再送信が発生した場合と従来手法を用いて曲面を生成し形状変化による再送信が発生した場合の描画比較を行った。ポリゴン数は 64×64 となっており、形状に変更を加えて再描画をする処理を 1 万回行い、1 秒あたりの描画回数を記録した。また、形状に変更がない場合の描画に関しても記録を行った。表 3.2 は記録した結果となっている。実験の結果から情報の再送信が発生した場合に従来の生成手法よりも速く処理を行うことができた。また、形状変更がない場合でも従来手法とほとんど変わりがない結果となった。

表 3.2　1 秒あたりの平均描画回数

本手法 (形状変更あり)	594.954 回/秒
従来手法 (形状変更あり)	192.901 回/秒
本手法 (形状変更なし)	596.623 回/秒
従来手法 (形状変更なし)	593.859 回/秒

3.4　結論と考察

3.3 節より、モデル情報の再送信が発生する場合の速度比較では従来手法より本手法のほうが早く処理を行うことができた。また、モデル情報の再送信が発生しない場合において、本手法と従来手法の描画速度はほぼ同じであった。そのため、速度比較の結果にて多くのモデルの情報の再送信が発生する場合に送信する情報量が少ない本手法の方が有用である。また、地形の形状について複数のパーリンノイズによる形状の表現を行い、生成する地形のサイズに合わせた表現を実現できた。

3.2 節より、地形生成のために 4 つのパーリンノイズの設定をおこなった。パーリンノイズの設定では、振幅や周波数による調整は想像しやすいが、実際に地形のどこの部分がどのような形状になるか実行してみるまで分かりにくいため、形状の調整には多くの時間がかかってしまう。また、本手法では地形の細かい形状をパーリンノイズを使用して表現を行っており、どのような形状になるかはノイズによって決まっている。そのため、地形の形状をある程度設定することが可能だが、地形の細かい部分を精密に生成したい場合には本手法は難しいという問題がある。そして、地形の色付けの実装では、色付け用のノイズと地形の高さに合わせていくつかの色を設定を行った。しかし、かなり狭い範囲の地形を生成する場合、場所によって色の塗り方バラバラになるので、色塗りに関して地形の場所や条件に合わせたいくつかの細かい設定をする必要になる。

第 4 章

まとめ

近年ではハードウェアの性能が上がり、広大な地形を移動し遊ぶことができるゲームが多く存在している。一般的に GPU を使用する場合はモデルの情報を一度のみ送信してメモリ上に保存し、メモリ上の情報を繰り返し使い表示を行う。モデルの形状データや頂点数が変わる場合にモデルの情報を再送信するが、再送信する情報量はなるべく減らす必要がある。

本論文では、生成する地形の領域や位置を変更したい場合、少ない情報の再送信によって、動的に地形の生成や変更が行えることを目的とした。パラメトリック曲面の一つである Gregory 曲面とテッセレーションシェーダ、パーリンノイズを使い再送信する情報量の少ない地形の生成方法を提案した。各曲面の領域情報と制御点情報を GPU に一度だけ送り、生成する地形の範囲情報を適宜再送信を行う。その後、テッセレーションシェーダと複数のパーリンノイズを使用して地形の形状を生成し、フラグメントシェーダで色付けをして、少ない情報の再送信による地形生成を行った。そして、広い範囲の地形生成から、いくら狭い範囲の地形生成を行った場合でも GPU に送信する情報量の変わらない地形生成が行えた。

モデルの形状を変更して再送信を発生させ、従来手法と本手法の描画速度の比較をした。比較を行った結果、本手法の描画速度が従来手法よりも早い結果となった。このことから、多くの情報の再送信が発生する場合に高速に描画処理が行えるという点で有効であることが分かった。また、情報の再送信が発生しない場合では、本手法と従来手法で描画速度にほとんど差が見られず、従来手法同様の描画速度で処理が行ることも分かった。したがって、多くの情報の再送信が発生する場合に送信する情報量が少ない本手法は有用である。

しかし、本手法では地形の細かい形状をパーリンノイズを使用して表現を行っており、地形の細かい部分を精密にデザインしたい場合に調整が難しい。また、色付けでは、生成する範囲によって色がまばらになってしまうため、もう少し地形の場所や条件に合わせた細かい設定をする必要になる。

謝辞

本研究を進めるために多くのご指導やアドバイスをしてくださった先生方や研究室のメンバーに感謝を述べたいと思います。

学部 1 年から修士 2 年までの 6 年間という長い間、研究のご指導や数多くの相談をしていただいた渡辺先生に感謝いたします。学部の時はプログラミングや数学の授業、先端メディア、学部での研究など多くの場面でお世話になりました。修士になってからは、紆余曲折あった研究のご指導や相談、学会投稿への添削や準備、そしてコロナ下での様々なサポートなど、この 6 年の間に色々な場面で多くのご指導をしていただいた渡辺先生に心から感謝しております。

学部 1 年から修士 2 年にかけて 6 年間、授業や演習、研究に関する助言をくださった三上先生に感謝いたします。学部から修士にかけて様々な授業をしていただき、特に演習に関してはとてつもなくお世話になりました。また、研究に関しては学部から修士までの長い間、色々な助言をしていただき真にありがとうございます。

学部から修士にかけて、色々な授業や研究に関する助言や指摘をくださった柿本先生に感謝いたします。授業では CG に関する様々なことを教えていただき、研究に関しては毎週のミーティングにて多くの助言やご指摘をしていただいたことに感謝いたします。

予備審査と最終審査にてご指摘をしていただいた羽田先生に感謝いたします。

学部 4 年から修士 2 年までの間、研究に関する助言や相談をしていただいた阿部先生に感謝いたします。特に何かの締め切り間際に多くのご指導や添削をしていただいたこと、とても感謝いたします。あのご指導や添削がなければ、きっと 1 つ 2 つ大きなミスをしていました。本当にありがとうございます。

また、卒業していった研究室のメンバーや留学生のみんな、そして一緒に苦楽をともにした修士のメンバーに感謝いたします。学部時代の研究室のメンバーには良い思い出を作らせてくれたこと、とても感謝しています。あの 1 年間は一番濃密で一番楽しい時間だった。落ち着いたらまた集まろう。

研究室にいた留学生のメンバーにはいい刺激をくれたことに感謝しています。まず、英語が少し話せるようになり、英語に対して苦手意識が無くなったのは君たちのおかげだ、ありがとう。そして、お互いの国の話をしたり、文化の違いで驚いたり、お互いの研究の相談をしたり、君たちに会ってから多くのいい経験ができた。会えなくなってしまったが、またいつか一緒にご飯でも食べに行こう。

卒業してしまった人も含め、修士のメンバーには一緒に戦ってくれたことを感謝しています。先生たちには多くのご指導やサポートをしてもらったけど、君たちがいなければ冗談抜きにどこかの段階で挫折していたと思う。つらい時も楽しい時も一緒に愚痴をこぼしたり、励ましあったり、腹から笑ったりしてくれる友がいたから最後まで進めました。本当にありがとう。

最後に色々と迷惑をかけ、手助けをしてくれた両親に感謝します。大学の 6 年間、生まれてからは 24 年間、ここまで何不自由なく生活をできたのは 2 人のおかげです。本当にありがとう。

参考文献

[1] SQUARE ENIX. Final fantasy 15. http://www.jp.square-enix.com/ff15/. 参照: 2020.11.21.

[2] Ubisoft Entertainment. Farcry5. https://ubisoft.co.jp/farcry5/. 参照: 2020.11.21.

[3] ソニー・インタラクティブエンタテインメント. Ghost of tsushima. https://www.playstation.com/ja-jp/games/ghost-of-tsushima/. 参照: 2020.11.29.

[4] Hisa Ando. GPU を支える技術超並列ハードウェアの快進撃 [技術基礎]. 株式会社技術評論社, 2017.

[5] MattPharr 著, 中本浩訳. *GPU Gems 2*. 株式会社ボーンデジタル, 2005.

[6] Jeremy Moore. Terrain rendering in 'far cry 5'. https://www.gdcvault.com/play/1025480/Terrain-Rendering-in-Far-Cry. 参照: 2020-11-20.

[7] 石川貴之, 青木由直. フラクタルによる図形の生成とその応用. テレビジョン学会技術報告, Vol. 8, No. 46, pp. 49–54, 1985.

[8] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Commun. ACM*, Vol. 25, No. 16, p. 371–384, 1982.

[9] 安居院猛, 宮田一乗, 中嶋正之. 三次元山岳形状の等高線からの自動作成法. 電子情報通信学会論文誌 D, Vol. 69, No. 12, pp. 1905–1912, 1986.

[10] 張志毅, 今野晃市, 徳山喜政. 周期的 B-Spline 曲線を利用した山岳地形の 3 次元モデル生成手法. 情報処理学会研究報告グラフィクスと CAD（CG）, No. 86, pp. 1–6, 2003.

[11] Losasso Frank and Hoppe Hugues. Geometry clipmaps: Terrain rendering using nested regular grids. *ACM SIGGRAPH 2004 Papers*, No. 8, pp. 769–776, 2004.

[12] L. M. Hwa, M. A. Duchaineau, and K. I. Joy. Real-time optimal adaptation for planetary geometry and texture: 4-8 tile hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 4, pp. 355–368, 2005.

[13] Zhang Jie, Zheng Changwen, Lv Pin, and Hu Xiaohui. Implicit restricted quadtree based visualization of large scale terrain. *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, No. 2, pp. 177–178, 2010.

[14] F.Strugar. Continuous distance-dependent level of detail for rendering heightmaps. *Journal of Graphics, GPU, and Game Tools*, Vol. 14, No. 4, pp. 57–74, 2009.

[15] Judnich John and Ling Nam. Fast multiresolution terrain rendering with symmetric cluster sets. *SIGGRAPH Asia 2011 Sketches*, Vol. 29, No. 2, pp. 1–2, 2011.

[16] Rui Zhai, Ke Lu, Weiguo Pan, and Shuangfeng Dai. Gpu-based real-time terrain rendering: Design and implementation. *Neurocomputing*, Vol. 171, pp. 1 – 8, 2016.

[17] HyeongYeop Kang, Yeram Sim, and JungHyun Han. Terrain rendering with unlimited detail and resolution. *Graphical Models*, Vol. 97, pp. 64–79, 2018.

[18] B. Santerre, M. Abe, and T. Watanabe. Improving gpu real-time wide terrain tessellation using the new mesh shader pipeline. pp. 86–89, 2020.

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

[20] P.Bézier. Definition numerique des courbes et surfaces. *Automatisme*, Vol. 11, , 1966.

[21] P.Bézier. Definition numerique des courbes et surfaces(ii). *Automatisme*, Vol. 12, , 1967.

[22] 千代倉 弘明鳥谷 浩志. 3 次元 CAD の基礎と応用. 共立出版, 1991.

[23] H.Chiyokura and F.Kimura. Design of solids with free-form surfaces. *Computer Graphics*, Vol. 17, No. 3, pp. 289–298, 1983.

[24] Mark Segal, Kurt Akeley, Jon Leech 著松田晃一, 内藤剛人, 竹内俊治・神田崇史訳. OpenGL4.0 グラフィックスシステム. 株式会社カットシステム, 2010.

[25] DavidWolff 著, 中本浩訳. OpenGL4.0 シェーディング言語. 株式会社ボーンデジタル, 2012.

[26] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, Vol. 19, No. 3, p.

287–296, 1985.

[27] Ken Perlin. Improving noise. *ACM Trans. Graph*, Vol. 21, No. 3, pp. 681–682.

[28] 安達翔平, 宇梶弘晃, 小坂昂大, 森島繁生. Perlin noise を用いた短繊維生成法による埃の高速描画手法. 情報処理学会研究報告グラフィクスと CAD（CG）, Vol. 2013-CG-150, No. 6, pp. 1–7, 2013.

[29] 間淵聡, 藤代一成, 大野義夫. 粒子ベース火炎レンダリング. 情報処理学会研究報告グラフィクスと CAD（CG）, Vol. 2011-CG-142, No. 21, pp. 1–6, 2011.

[30] Fine Kernel Project. Fine kernel tool kit. `https://gamescience.jp/FK`. 参照: 2021-2-6.

発表業績

ポスター発表

1. 山本馨加, 阿部雅樹, 渡辺大地, 形状変化に対応したトリム曲面の生成, 第 19 回ビジュアル情報処理研究合宿, 2019.
2. 山本馨加, 阿部雅樹, 渡辺大地, カメラ距離に応じたテッセレーションシェーダによるモデルの細分割とテクスチャの動的生成,　映像表現・芸術科学フォーラム, 2020.
3. 山本馨加, 阿部雅樹, 渡辺大地, カメラ距離に応じた多重解像度の地形生成,　NICOGRAPH2020, 2020.

口頭発表

1. 山本馨加, 阿部雅樹, 渡辺大地, テッセレーションシェーダを用いた描画要素数の動的制御による地形描画高速化に関する研究,　第 27 回デジタルコンテンツクリエーション合同研究発表会, 2021.