Vol.2021-GN-112 No.16
Vol.2021-CDS-30 No.16
Vol.2021-DCC-27 No.16
2021/1/25

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

山本 馨加[1]　阿部 雅樹[†1]　渡辺 大地[†1]

概要：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 を使用し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。しかし、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 と GPU 間での多くの情報のやり取りが発生して処理が重くなってしまうため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 Gregroy 曲面とテッセレーションシェーダを使い、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、適宜生成する地形の範囲を送信することで、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生成が行えた。また、最後に周波数や振幅の違う複数パーリンノイズを合成して使用し、生成する地形の範囲に対応した細かい形状の表示も行うことができた。

キーワード：Tessellation Shader、Gregory 曲面、地形生成、GPU、パーリンノイズ

1. はじめに

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、ゲーム画面内に多数の建物やキャラクターといった 3D モデルを表示することが可能となった。FINAL-FANTASY15[1] や FARCRY5[2] や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。本研究はテッセレーションシェーダを用いた地形生成の高速描画を提案する。ハードウェア性能の向上により、ゲーム内では多くの 3D モデルの表示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。一般的にゲームでは描画処理を減らす方法として Level Of Detail（以降「LOD」）がある。LOD は、事前にポリゴン数の異なる 3D モデルを複数パターン用意しておき、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手法である。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処理を軽くしている。

GPU 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 内のメモリ上に送ってき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。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、CPU から GPU への情報多くの情報を再送信すると処理が遅くなってしまう。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

[1] 東京工科大学大学院バイオ・情報メディア研究科
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，東京工科大学メディア学部
Presently with Tokyo

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おいたものを 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[3] では、初めに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 内で何キロもの平たく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。次にカメラ位置を算出し、先ほど分割し保存して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。この時、多少カメラが移動することを想定し、選ぶ分割段階の地形をある程度広く取って選択する。分割した地形と選んだ分割段階に対応するテクスチャ情報を GPU に送る。そして、GPU に送った情報をもとに、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、さらなる分割および分割結果の保存、対応テクスチャの準備が必要となる。表示する地形が細かいほど GPU へ多くの情報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 [4] や中点変位法を用いた手法 [5] などのフラクタル手法を使ったものがある。また、GPU を用いた地形生成として Losasso らの研究 [6] や Hwa らの研究 [7] 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 らの研究 [8] がある。事前に地形を分割して保持しておき、LOD 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 LOD の情報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 らの研究 [9] 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、HyeongYeop らの研究でも事前に地形を分割して保持しておき、必要な詳細の地形を 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と多くの情報の再送信が発生する。

本研究では、3 種類の初期情報と 1 種類の更新情報により、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。生成する地形の領域や位置を変更したい場合、少数の更新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。また、表示する領域に合わせて地形の形状も変化させ、広い領域の地形を表示した場合には大まかな地形の形状を主に表示し、狭い領域の地形を表示する場合には、広い領域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 に送る初期情報は、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 種類である。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 に一度だけ送信する。更新情報は生成する地形の範囲であり、生成地形の変更のたびに GPU へ再送信する。これらの送信情報をもとに 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。従来手法では事前の地形分割処理や、カメラ位置に応じた GPU への多量な地形情報転送が必要なのに対し、本手法では制御点の情報と各曲面の領域情報をもとに GPU 内で生成する地形の範囲情報を使い地形の生成を行っている。カメラ距離が近づいて、狭い領域を表示したい場合でも生成する地形の範囲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 内で地形生成をしているため地形の大小に関わらず、送信情報量は同量のまま同一頂点数で曲面を生成することが可能である。地形の形状表現にはパーリンノイズを使用し、複数のパーリンノイズを合成することで生成する地形の領域に対応した詳細な地形の形状の生成を行った。

2. 提案手法

本研究では、パラメトリック曲面とテッセレーションシェーダを用いて、少ない情報の再送信により、地形の生成を行う。初めに 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。

3. 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[10] 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[11] 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の制御点を GPU に送り、送った制御点情報から GPU 内で曲面形状の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。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、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。パラメトリック曲面として Gregory 曲面 [12] を選択した。

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 (1) のようになる。

$$\mathbf{S}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)\mathbf{P}_{ij} \tag{1}$$

$\mathbf{P}_{ij}$ 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) は Bernstein 基底関数をあらわしたものである。

$$B_i^n(t)=\left(\begin{array}{c} n \\ i \end{array}\right)t^i(1-t)^{n-i} \tag{2}$$

ここで式 (3) は 2 項係数である.

$$\left(\begin{array}{c} n \\ i \end{array}\right)={}_nC_i=\frac{n!}{i!(n-i)!} \tag{3}$$

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 $\mathbf{P}_{ijk}(i=0\ldots3,j=0\ldots3,k=0,1)$ にて表す。式 (4) は Gregory 曲面の曲面表現式を表したものである。

$$\mathbf{S}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)\mathbf{Q}_{ij}(u,v) \tag{4}$$

$B_i^n(u)$ と $B_j^m(v)$ は式 (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、次のような関係がある。

•$i \neq$1,2 または $j \neq$1,2 のとき

$$\mathbf{Q}_{ij}=\mathbf{P}_{ij0} \tag{5}$$

•$i=$ 1,2 かつ $j=$ 1,2 のとき

$$\begin{aligned}
\mathbf{Q}_{11}(u,v)&=\frac{u\mathbf{P}_{110}+v\mathbf{P}_{111}}{u+v}\\
\mathbf{Q}_{12}(u,v)&=\frac{u\mathbf{P}_{120}+(1-v)\mathbf{P}_{121}}{u+(1-v)}\\
\mathbf{Q}_{21}(u,v)&=\frac{(1-u)\mathbf{P}_{210}+v\mathbf{P}_{211}}{(1-u)+v}\\
\mathbf{Q}_{22}(u,v)&=\frac{(1-u)\mathbf{P}_{220}+(1-v)\mathbf{P}_{221}}{(1-u)+(1-v)}
\end{aligned} \tag{6}$$

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 $\mathbf{A}_i(i=0\ldots3),\mathbf{B}_i(i=0\ldots3),\mathbf{C}_i(i=0,1,2)$ とすると、

$$\mathbf{A}_1=\frac{2\mathbf{A}_0+\mathbf{A}_3}{3},\mathbf{A}_2=\frac{\mathbf{A}_0+2\mathbf{A}_3}{3} \tag{7}$$

と仮定した場合、式 (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。

$$\begin{aligned}
\mathbf{B}_1&=\frac{(k_1-k_0)\mathbf{A}_0+3k_0\mathbf{A}_1+2h_0\mathbf{C}_1+h_1\mathbf{C}_0}{3},\\
\mathbf{B}_2&=\frac{3k_1\mathbf{A}_2-(k_1-k_0)\mathbf{A}_3+h_0C_2+2h_1\mathbf{C}_1}{3}
\end{aligned} \tag{8}$$

ただし、k_0,k_1,k_2,k_3 は

$$\begin{aligned}
\mathbf{B}_0&=k_0\mathbf{A}_0+h_0\mathbf{C}_0,\\
\mathbf{B}_3&=k_1\mathbf{A}_3+h_1\mathbf{C}_2
\end{aligned} \tag{9}$$

を満たす実数である。

4. 各曲面の領域情報

各曲面の領域情報は各 Gregroy 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 Gregroy 曲面ごとにこれを設定していく。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 Gregroy 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 Gregroy 曲面の制御点と各 Gregroy 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 1 は、最小値と最大値を

Vol.2021-GN-112 No.16
Vol.2021-CDS-30 No.16
Vol.2021-DCC-27 No.16
2021/1/25

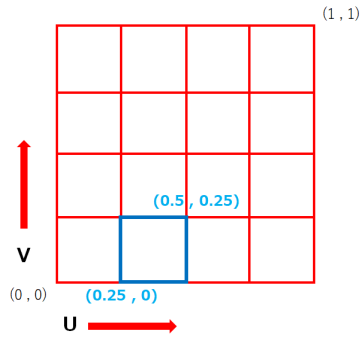

図 1 各曲面の最小値と最大値の設定

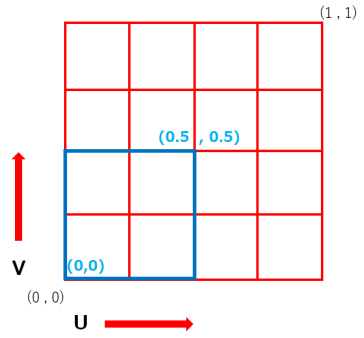

図 2 生成する地形の範囲の設定

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 番目の青い枠線が最小値 a_{ij} を 0.25、b_{ij} を 0、最大値 e_{ij} を 0.5、f_{ij} を 0.25 の設定を行った曲面であり、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いく。

5. 地形生成

5.1 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た場合の表示領域を決定する情報である。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、地形全体の左下 1/4 の部分を表示したい場合、UV 座標系で最小値を c、d と最大値を g、h として、c は 0、d は 0 となり、g は 0.5、h は 0.5 となる。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 は全体の左下 1/4 の部分を表示したい場合の設定様子の図であり、青い領域が設定した生成する地形の範囲となる。

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、生成する地形の範囲情報を再送信するのみで、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できる。

5.2 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。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 Gregroy 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。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 Gregroy 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。

5.3 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 [13] 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。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している。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る。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

Vol.2021-GN-112 No.16
Vol.2021-CDS-30 No.16
Vol.2021-DCC-27 No.16
2021/1/25

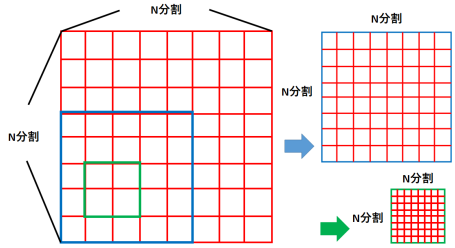

図 3 生成する曲面のサイズが変更した場合の頂点の例

ら最終的な頂点の位置を計算して決定する。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 3 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。

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 内で各 Gregroy 曲面の形状を式 (4) で計算する。式 (4) で使用する u,v は GPU に送信した情報をもとに計算を行う。u,v を求める式を表したものが式 (10) であり、u と v に入る値は 0~1 を超えなようにする。gl_TessCoord で取得した x 座標が o、y 座標が p となっている。

$$u = (c - a_{ij})/(e_{ij} - a_{ij}) + o(1 - (g - e_{ij})/(a_{ij} - e_{ij}))$$
$$v = (d - b_{ij})/(f_{ij} - b_{ij}) + p(1 - (h - f_{ij})/(b_{ij} - f_{ij})) \tag{10}$$

式 (10) で求めた u と v を式 (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 [14][15] 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。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 (11) はパーリンノイズに使用する値を計算したものである。

$$q = a_{ij} + u(e_{ij} - a_{ij})$$
$$r = b_{ij} + v(f_{ij} - b_{ij}) \tag{11}$$

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 (12) は 2 つのパーリンノイズを組み合わせた式である。

$$p = K(q, r) + \frac{K(\alpha q, \alpha r)}{\beta} \tag{12}$$

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 (12) で求めた値を式 (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 (13) は最終的な形状の高さ方向の計算式である。

$$S'_z = S_z + p \tag{13}$$

地形の色はフラグメントシェーダ内で色を設定する。色をつけるためのパーリンノイズをテッセレーションシェーダ内で用意して、頂点位置をパーリンノイズに与えて数値を算出する。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 で設定したものが C であり、高さに応じて塗る別の色の RGB が B となる。高さに応じて C と B 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 (14) 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 (15) で最終的な色を決定する。

$$s = l + \frac{S'_z}{\gamma} \tag{14}$$

$$A = Bs + C(1 - s) \tag{15}$$

6. 出力結果と検証

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合成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の c を 0、d を 0 と設定し、最大値である g と h を $g = 1, h = 1$ から 0.05 ずつ減少させて生成した地形の

Vol.2021-GN-112 No.16
Vol.2021-CDS-30 No.16
Vol.2021-DCC-27 No.16
2021/1/25

実行結果を示す。以下の図 4 から図 22 は最大値を 0.05 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。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、広い範囲の地形から狭い範囲の地形の生成の変化の様子がわかる。

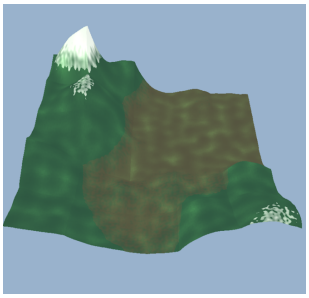

図 4 g と h が 1.0 の地形

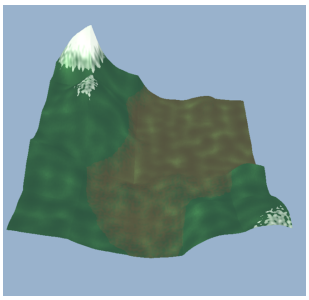

図 5 g と h が 0.95 の地形

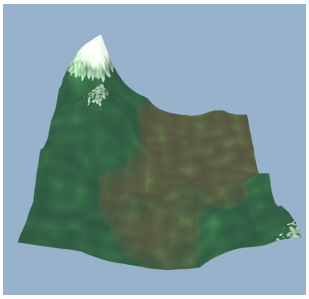

図 6 g と h が 0.9 の地形

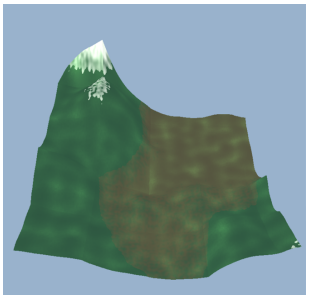

図 7 g と h が 0.85 の地形

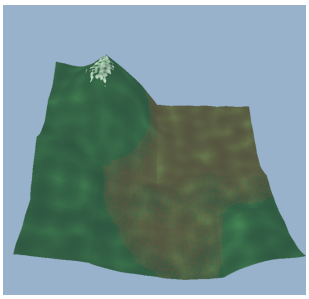

図 8 g と h が 0.8 の地形

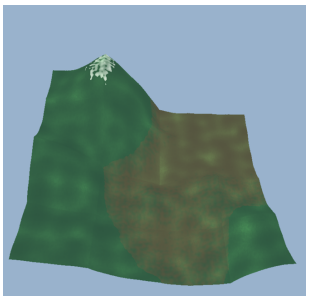

図 9 g と h が 0.75 の地形

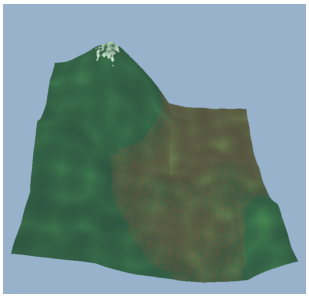

図 10 g と h が 0.7 の地形

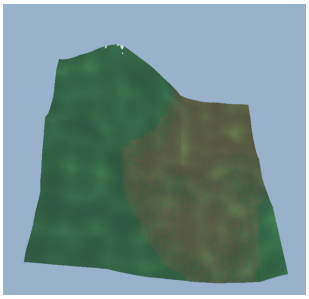

図 11 g と h が 0.65 の地形

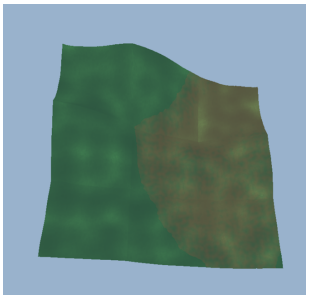

図 12 g と h が 0.6 の地形

以下の図 23 は一枚の曲面を大きく生成した場合と小

Vol.2021-GN-112 No.16
Vol.2021-CDS-30 No.16
Vol.2021-DCC-27 No.16
2021/1/25

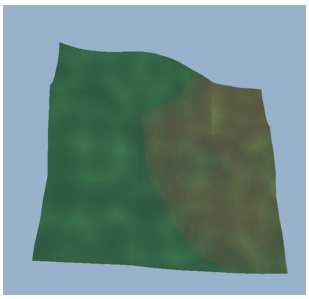

図 13 g と h が 0.55 の地形

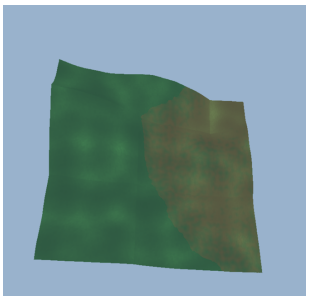

図 14 g と h が 0.5 の地形

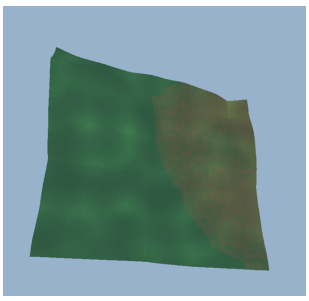

図 15 g と h が 0.45 の地形

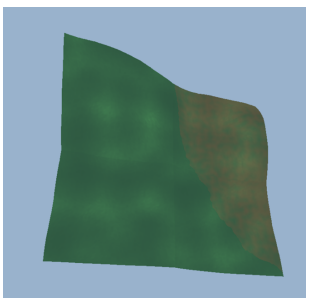

図 16 g と h が 0.4 の地形

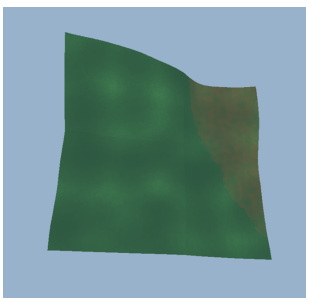

図 17 g と h が 0.35 の地形

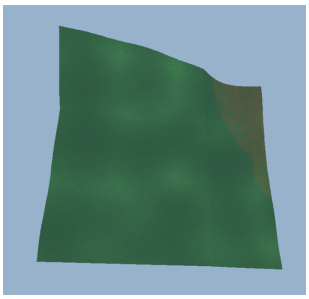

図 18 g と h が 0.3 の地形

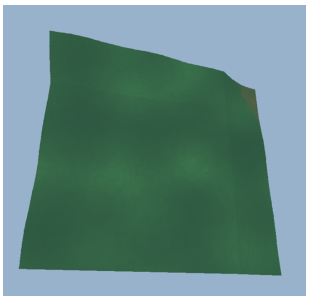

図 19 g と h が 0.25 の地形

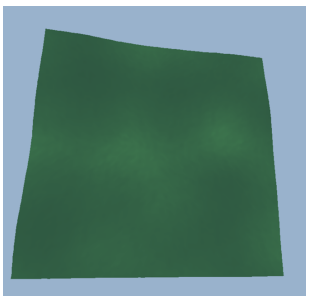

図 20 g と h が 0.2 の地形

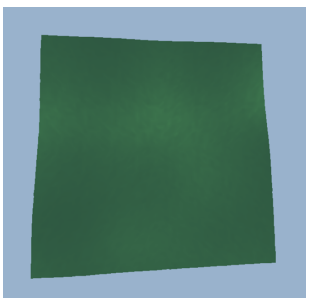

図 21 g と h が 0.15 の地形

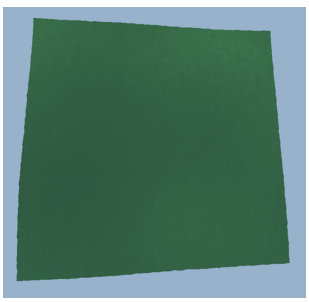

図 22 g と h が 0.1 の地形

Vol.2021-GN-112 No.16
Vol.2021-CDS-30 No.16
Vol.2021-DCC-27 No.16
2021/1/25

さく生成した場合に頂点数を示した図である。最小値を $c = 0, d = 0$ と設定し、上の 2 つが最大値 $g = 0.2, h = 0.2$ で地形生成を行った場合であり、下の 2 つが最大値 $g = 0.02, h = 0.02$ で地形生成を行った場合の頂点数を示したものである。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。生成する範囲が小さくなっても同じ頂点数になっている。そのため、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可能である。また、パーリンノイズを使い地形に加えた細かい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。

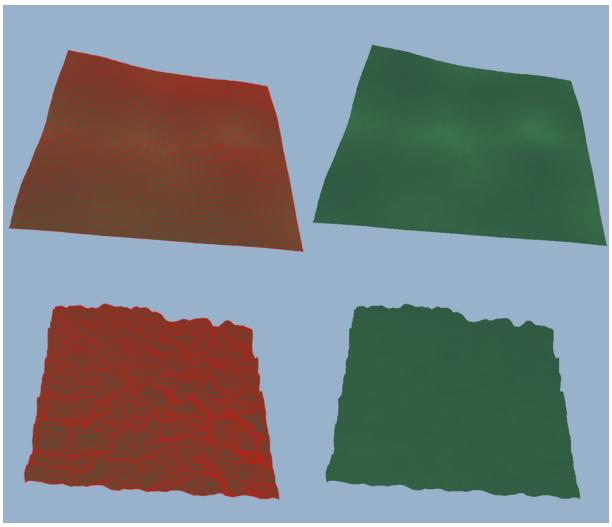

図 23 g と h が 0.2 の地形と g と h が 0.02 の地形の生成比較

一枚の曲面を作成して一度 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を再送信した場合の速度の検証のために本手法を用いて一枚の曲面を生成して、生成した形状に変更を加えて描画を行う処理を本手法を用いた場合と変更があった箇所の形状情報を再送信した場合の比較を行った。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 1 は記録した結果となっている。実験に用いた PC のスペックを表 2 を示す。

表 1 1 秒あたりの平均描画回数

本手法	594.954
変更箇所の再送信	192.901
形状変更なし	596.623

表 2 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

7. まとめ

本論文では、パラメトリック曲面の一つである Gregroy 曲面とテッセレーションシェーダを使い、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、生成する地形の範囲を再送信することで、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地形生成が行えた。また、複数のパーリンノイズを合成して生成する地形の範囲に対応した、細かい形状の表示を行うことができた。

参考文献

[1] FINALFANTASY15: FINAL FANTASY XV （ファイナルファンタジー 15） — SQUARE ENIX, SQUARE ENIX（オンライン），入手先 ⟨http://www.jp.square-enix.com/ff15/⟩（参照 2020-11-21）.
[2] FARCRY5: Far Cry & reg;5 - ファークライ 5 — トップページ — Ubisoft, Ubisoft Entertainment（オンライン），入手先 ⟨https://ubisoft.co.jp/farcry5/⟩（参照 2020-11-21）.
[3] Moore, J.: Terrain Rendering in 'Far Cry 5', Ubisoft Entertainment (online), available from ⟨https://www.gdcvault.com/play/1025480/Terrain-Rendering-in-Far-Cry⟩ (accessed 2020-11-20).
[4] 青木 由直石川 貴之：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).
[5] Fournier, A., Fussell, D. and Carpenter, L.: Computer Rendering of Stochastic Models, *Commun. ACM*, Vol. 25, No. 16, p. 371–384 (1982).
[6] Losasso, F. and Hoppe, H.: Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids, *ACM SIGGRAPH 2004 Papers*, No. 8, pp. 769–776 (2004).
[7] Hwa, L. M., Duchaineau, M. A. and Joy, K. I.: 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).
[8] Zhai, R., Lu, K., Pan, W. and Dai, S.: GPU-based real-time terrain rendering: Design and implementation, *Neurocomputing*, Vol. 171, pp. 1 – 8 (2016).
[9] Kang, H., Sim, Y. and Han, J.: Terrain rendering with unlimited detail and resolution, *Graphical Models*, Vol. 97, pp. 64–79 (2018).
[10] S.A.Coons: Surfaces for computer-aided design of space figures, *MIT* (1964).
[11] P.Bézier: Definition numerique des courbes et surfaces, *Automatisme*, Vol. 11 (1966).
[12] 千代倉 弘明鳥谷 浩志：3 次元 CAD の基礎と応用，共立出版 (1991).
[13] Segal, M., Akeley, K., 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).
[14] Perlin, K.: An Image Synthesizer, *SIGGRAPH Comput. Graph.*, Vol. 19, No. 3, p. 287–296 (1985).
[15] Perlin, K.: Improving Noise, *ACM Trans. Graph*, Vol. 21, No. 3, pp. 681–682.