コンピュータグラフィクス論　モデリング課題

05-242628 三田村彰大（理学部地球惑星物理学科）

動作が確認できる URL：　https://shota-mitamura.com/works/computergraphicskadai1/template
コード置き場：　https://github.com/ramutami/modeling_mitamura
AI とのチャットログ：　https://chatgpt.com/g/g-p-69fa...

1　概要

パラメトリック曲線、特に Catmull-Rom スプラインを、パラメタが Uniform な場合と Centripetal な場合で実装し、両者を比較する。特に、Catmull-Rom スプラインが制御点を滑らかに補間することや、Centripetal なパラメタ区分を用いることで望ましくない挙動を抑制できることなどを確認する。

※AI の利用について

本レポートでは、パラメトリック曲線の実装にあたり.html/.css/.js ファイルの雛形を AI に相談しながら作成した[*1]。具体的には、動作確認用の Website の見た目や、制御点のドラッグ操作、スライダによる制御点の個数変更などの UI の実装周りを ChatGPT を用いたバイブコーディングで行った。その上で実際のパラメトリック曲線の実装やパラメタ区分の実装は自分で行うこととした[*2]。

2　理論

2.1　Catmull-Rom スプライン

時刻 t_k での値が $\mathbf{x}_k$ で表されるような状況を考える。(Figure1参照) この時、$\mathbf{x}_k$ と $\mathbf{x}_{k+1}$ を繋ぐ曲線 $r_k(t)$ (図中では $x_k(t)$ として表示) を $\mathbf{x}_{k-1}, \mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{x}_{k+2}$ を用いて次のように決定する。

1. 直線による補間を行う。(Figure2参照)
   - $\mathbf{x}_{k-1} \sim \mathbf{x}_k$ を結ぶ直線 $l_{k-1}(t)$ を求める。
   - $\mathbf{x}_k \sim \mathbf{x}_{k+1}$ を結ぶ直線 $l_k(t)$ を求める。
   - $\mathbf{x}_{k+1} \sim \mathbf{x}_{k+2}$ を結ぶ直線 $l_{k+1}(t)$ を求める。
2. 二次曲線による補間を行う。(Figure3参照)
   - l_{k-1} と l_k を用いて、$\mathbf{x}_{k-1} \sim \mathbf{x}_k \sim \mathbf{x}_{k+1}$ の間を二次曲線 $q_k(t)$ で補間する。
   - l_k と l_{k+1} を用いて、$\mathbf{x}_k \sim \mathbf{x}_{k+1} \sim \mathbf{x}_{k+2}$ の間を二次曲線 $q_{k+1}(t)$ で補間する。
3. 三次曲線による補間を行う。(Figure4参照)
   - q_k と q_{k+1} を用いて、$\mathbf{x}_k \sim \mathbf{x}_{k+1}$ の間を三次曲線 $r_k(t)$ で補間する。

ただし、それぞれの曲線（直線）の式は次のように定める。

$$l_k(t) = \left(1 - \frac{t - t_k}{t_{k+1} - t_k}\right)\mathbf{x}_k + \frac{t - t_k}{t_{k+1} - t_k}\mathbf{x}_{k+1} \tag{1}$$

$$q_k(t) = \left(1 - \frac{t - t_{k-1}}{t_{k+1} - t_{k-1}}\right) l_{k-1}(t) + \frac{t - t_{k-1}}{t_{k+1} - t_{k-1}} l_k(t) \tag{2}$$

$$r_k(t) = \left(1 - \frac{t - t_k}{t_{k+1} - t_k}\right) q_k(t) + \frac{t - t_k}{t_{k+1} - t_k} q_{k+1}(t) \tag{3}$$

よって実際に実装する上では、$\mathbf{x}_{k-1}, \mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{x}_{k+2}$ 及び $t_{k-1}, t_k, t_{k+1}, t_{k+2}$ 及び t を入力するとパラメトリック曲線上の座標 $r_k(t)$ を返すような次のような関数を定義する[*3]。

$$\begin{aligned}&\text{function}(t, \mathbf{x}_{k-1}, \mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{x}_{k+2}, t_{k-1}, t_k, t_{k+1}, t_{k+2})\{\\ &\quad \text{calculate: } l_{k_i} \text{ using } t, t_{k_i}, \mathbf{x}_{k_i}\\ &\quad \text{calculate: } q_{k_i} \text{ using } t, t_{k_i}, l_{k_i}\\ &\quad \text{calculate: } r_k \text{ using } t, t_{k_i}, q_{k_i}\\ &\quad \text{return: } r_k(t\,;\, \mathbf{x}_{k-1}, \mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{x}_{k+2}, t_{k-1}, t_k, t_{k+1}, t_{k+2})\\ &\}\end{aligned} \tag{4}$$

[*1] 普段 FORTRAN しか触っていないので…
[*2] AI の察しが良すぎたがために、UI の相談をしているだけであるにもかかわらずパラメトリック曲線の実装コードをサジェストされてしまったが、できるだけそれを無視してコーディングを行った。
[*3] 描画は二次元空間上に行うが、実装上は $z = 0$ とした三次元空間内で描画するので、実際の関数としては Three.Vector3(x,y,z) を返すものとなる。(一応コードでは、今後の拡張も考えて z 座標の更新もできるようになっているが、結局使わなかった…)

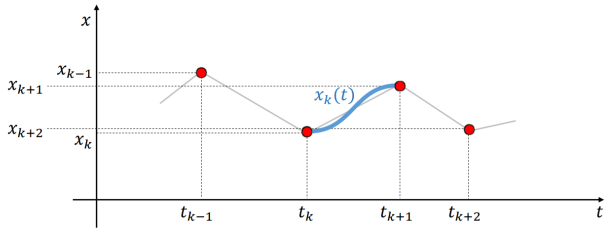

Figure1: 考えている問題設定。授業スライドより抜粋。

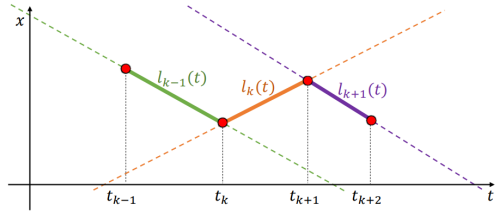

Figure2: 直線による補間。授業スライドより抜粋。

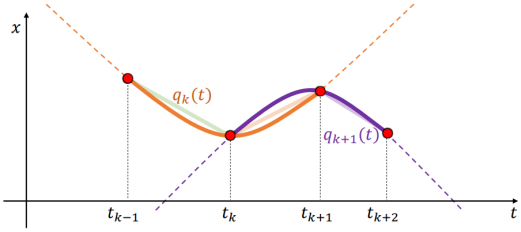

Figure3: 二次曲線による補間。授業スライドより抜粋。

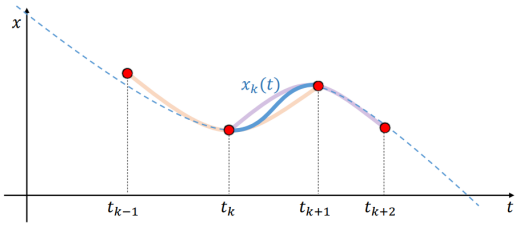

Figure4: 三次曲線による補間。ただし文中では $x_k(t)$ ではなく $r_k(t)$ としている。授業スライドより抜粋。

3 実装

3.1 Uniform なパラメタ区分

$t_k = t_{k-1} + 1$ で表されるような Uniform なパラメタ区分を考える。このとき、一般に $t_k = 0$ としても問題なく、

$$t_{k-1} = -1\ ,\ t_k = 0\ ,\ t_{k+1} = 1\ ,\ t_{k+2} = 2 \qquad (5)$$

を (1)〜(3) に代入しながら先のアルゴリズムに従って $r_k(t)$ を決定すると最終的なパラメタ曲線の座標は次のように具体的に計算することができる。

$$\begin{aligned}
&r_k(t\,;\,\mathbf{x}_{k-1},\mathbf{x}_k,\mathbf{x}_{k+1},\mathbf{x}_{k+2}) \\
&= (1-t)\left[\left(1-\frac{t+1}{2}\right)\left\{(1-(t+1))\,\mathbf{x}_{k-1}+(t+1)\,\mathbf{x}_k\right\}\right. \\
&\qquad \left.+\left(\frac{t+1}{2}\right)\left\{(1-t)\,\mathbf{x}_k+t\,\mathbf{x}_{k+1}\right\}\right] \\
&\quad +t\left[\left(1-\frac{t}{2}\right)\left\{(1-t)\,\mathbf{x}_k+t\,\mathbf{x}_{k+1}\right\}\right. \\
&\qquad \left.+\frac{t}{2}\left\{(1-(t-1))\,\mathbf{x}_{k+1}+(t-1)\,\mathbf{x}_{k+2}\right\}\right] \qquad (6)
\end{aligned}$$

これはさらに整理することができて、Wikipediaに記載されているものと同様の次の式を得ることができる。

$$r_k(t) = \frac{1}{2}\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{x}_{k-1} \\ \mathbf{x}_k \\ \mathbf{x}_{k+1} \\ \mathbf{x}_{k+2} \end{bmatrix} \qquad (7)$$

これを実装したのが、catmuluniform.js 内の関数 catmull-Romuniform である。[*4]

catmullRomuniform 関数

```
function catmullRomuniform(pkm1, pk, pkp1, pkp2, t
    ) {
  const t2 = t * t;
  const t3 = t2 * t;

  const x = 0.5*(
    t3*(-1*pkm1.x+3*pk.x-3*pkp1.x+pkp2.x)+
    t2*(2*pkm1.x-5*pk.x+4*pkp1.x-pkp2.x)+
    t*(-pkm1.x+pkp1.x)+
    2*pk.x
  );

  const y = ...
  const z = ...

  return new THREE.Vector3(x, y, z);
}
```

よって points[ ] で表されるような制御点 $\mathbf{x}_0, \mathbf{x}_1, \ldots$ に対し、実際にパラメトリック曲線を描画する部分は次のようすれば良いとわかる。

パラメトリック曲線の描画部分

```
for (let i = 0; i < points.length - 1; i++) {
  const pk_1 = points[i-1];
  const pk = points[i];
  const pkp1 = points[i+1];
  const pkp2 = points[i+2];
  for (let j = 0; j < samplesPerSegment; j++) {
    const t = j / samplesPerSegment;
    curvePoints.push(catmullRomuniform(pk_1, pk,
        pkp1, pkp2, t));
  }
}
```

[*4] コード 57 行目。

すなわち制御点 $\mathbf{x}_{i-1} \sim \mathbf{x}_{i+2}$ を用いてパラメトリック曲線を計算し、時刻 $t = 0 + j\Delta t$ での曲線の座標を計算すれば良い。[*5]

3.2 端点処理

最初の制御点 $\mathbf{x}_0$ と $\mathbf{x}_1$ の間のパラメトリック曲線を描画する際、すなわち $\mathbf{x}_k = \mathbf{x}_0$ の時は、$\mathbf{x}_{k-1}$ として用いることのできる制御点が存在しない。そこでこの場合は、$\mathbf{x}_{k-1} = \mathbf{x}_k = \mathbf{x}_0$ として処理することにする。[*6]

これはすなわち、$\mathbf{x}_0$ と同じ値を持つ仮想的な制御点 $\mathbf{x}_{-1}$ を追加していることに他ならず、以降の計算アルゴリズムは変更を受けない。(Figure5参照。)(強いて言えば $(t, \mathbf{x})$ 空間で l_{k-1} 直線の傾きが平坦になるだけである。)

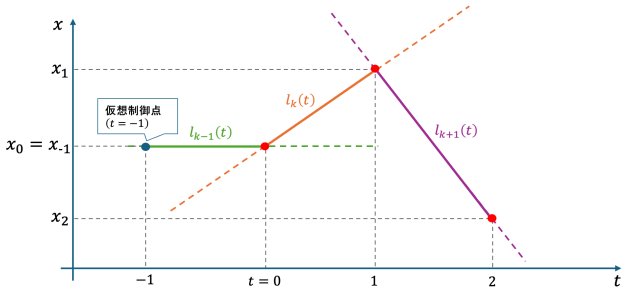

Figure5: ユニフォームなパラメタ区分における端点処理。

3.3 実装結果(Uniform なパラメタ区分)

ユニフォームなパラメタ区分を用いた場合の Catmull-Rom 曲線を描画した図が Figure6及び Figure7である。まず Figure6を見ると、全体的に滑らか(C^1 連続)な曲線となっており、端点付近でもその滑らかさが失われていないことがわかる。また、図中央の制御点が非常に密になっている箇所では、「くるりん」とでも言うべき望ましくない挙動 (ループ) が現れている。この挙動は後ほど Centripetal なパラメタ区分で解消されることが期待される。さらに、図右側中央の制御点が四角形を形成している箇所に着目すると、この場所でパラメトリック曲線は「ぼわっと」膨らんだ四角形を描いている。このような挙動を見ると、Catmull-Rom スプラインがシャープな角の表現を苦手としていることが推察される。

また、Figure7はネコの絵を Catmull-Rom 曲線で描いたものである。制御点をパラメトリック曲線が必ず通るため、非常に直感的かつ簡単な操作で絵を描くことができた。ただしこの場合においても、C^1 連続性を保つと言う要請が逆にシャープな角の描画を困難にしており、ネコの「耳」の接続部で不自然な凹みが生じてしまっていることがわかる。この点は centripetal で改善されるかどうかも注目する必要がある。

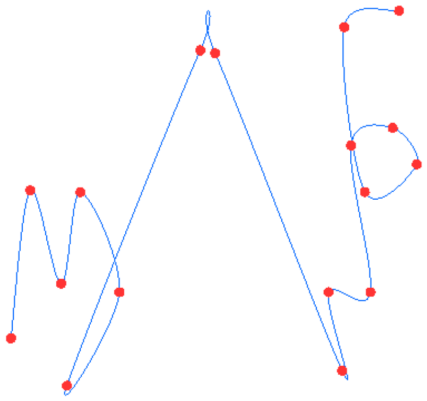

Figure6: ユニフォームなパラメタ区分を用いた Catmull-Rom 曲線。

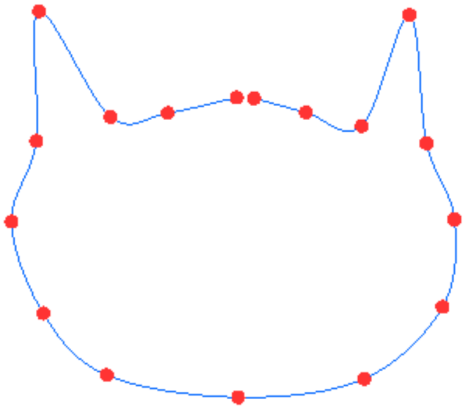

Figure7: ネコ (?) の絵。

3.4 Centripetal なパラメタ区分

次に、centripetal なパラメタ区分、すなわち $t_k = t_{k-1} + \sqrt{\|\mathbf{x}_{k-1} - \mathbf{x}_k\|}$ と表されるような場合に Catmull-Rom 曲線を実装することを考える。この場合、Uniform なパラメタ区分で行ったように $t_{k-1}, t_k, t_{k+1}, t_{k+2} = -1, 0, 1, 2$ を用いて計算を簡略化することはできず、愚直に (4) で示したような逐次的に $r_k(t)$ を計算する関数を定義する必要がある。(ただし、$t_k = 0$ は用いることにする。) それを実装したのが catmulcentripetal.js 内の catmullRomcentripetal 関数[*7]である。

catmullRomcentripetal 関数

```
function catmullRomcentripetal(pkm1, pk, pkp1,
    pkp2, tkm1,tkp1,tkp2,t) {
    // lkの計算(x座標)
    const lkm1x = (1-(t-tkm1)/(-tkm1))*pkm1.x + ((
        t-tkm1)/(-tkm1))*pk.x;
    const lkx = (1-t/tkp1)*pk.x + (t/tkp1)*pkp1.x;
    const lkp1x = (1-(t-tkp1)/(tkp2-tkp1))*pkp1.x
        + ((t-tkp1)/(tkp2-tkp1))*pkp2.x;
```

[*5] $t_i = 0$ としたことを利用している。
[*6] 最後の制御点 $\mathbf{x}_{N-1}$ と $\mathbf{x}_N$ の間のパラメトリック曲線の描画についても同様に処理する。
[*7] コード 58 行目

```
    //qkの計算(x座標)
    const qkx = (1-(t-tkm1)/(tkp1-tkm1))*lkm1x+ ((
        t-tkm1)/(tkp1-tkm1))*lkx;
    const qkp1x = (1-t/tkp2)*lkx+ (t/tkp2)*lkp1x;
    //rkの計算(x座標)
    const x = (1-t/tkp1)*qkx + (t/tkp1)*qkp1x;
    ...
    const y = ...
    const z = ...

  return new THREE.Vector3(x, y, z);
}
```

次に、この catmullRomcentripetal 関数を用いて実際にパラメトリック曲線を描画していくわけであるが、その際制御点の座標から $t_{k-1}, t_{k+1}, t_{k+2}$ の値を次のように計算する必要がある。[*8]

$$t_{k-1} = 0 - \sqrt{\|\mathbf{x}_{k-1} - \mathbf{x}_k\|} \tag{8}$$

$$t_{k+1} = 0 + \sqrt{\|\mathbf{x}_k - \mathbf{x}_{k+1}\|} \tag{9}$$

$$t_{k+2} = t_{k+1} + \sqrt{\|\mathbf{x}_{k+1} - \mathbf{x}_{k+2}\|} \tag{10}$$

また、実際にパラメトリック曲線を描画する上では、時刻 t のグリッド幅 Δt をあらかじめ定めておき、$t_k = 0 \sim t_{k+1}$ の間を $t_{k+1}/\Delta t$ 個のメッシュに区切って描画する。こうすることで、制御点間の距離が短い時は柔軟にパラメタの間隔を小さくしループなどの望ましくない挙動を抑えることができると期待される。

すなわち、実際にパラメトリック曲線を描画する部分は次のようになる。

loop for k: {
  calculate: $t_{k-1} = 0 - \sqrt{\|\mathbf{x}_{k-1} - \mathbf{x}_k\|}$
  calculate: $t_{k+1} = 0 + \sqrt{\|\mathbf{x}_k - \mathbf{x}_{k+1}\|}$
  calculate: $t_{k+2} = t_{k+1} + \sqrt{\|\mathbf{x}_{k+1} - \mathbf{x}_{k+2}\|}$
  loop for $t \in [0, \Delta t, 2\Delta t, \ldots, t_{k+1}]$ {
    add new point on line
      $\leftarrow r_k(t\,;\, \mathbf{x}_{k-1}, \mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{x}_{k+2}, t_{k-1}, t_k = 0, t_{k+1}, t_{k+2})$
  }
} (11)

(ただし r_k として先に示した catmullRomcentripetal 関数を用いることになる。)

これを実際に実装すると次のよう。

パラメトリック曲線 (centripetal) の描画部分

```
for (let i = 0; i < points.length - 1; i++) {
  const pkm1 = points[i-1];
  const pk = points[i];
  const pkp1 = points[i+1];
  const pkp2 = points[i+2];

  const tkm1 = -Math.sqrt(Math.sqrt((pkm1.x-pk.x)
      **2 + (pkm1.y-pk.y)**2 + (pkm1.z-pk.z)**2 ))
  const tkp1 = Math.sqrt(Math.sqrt((pkp1.x-pk.x)
      **2 + (pkp1.y-pk.y)**2 + (pkp1.z-pk.z)**2 ))
  const tkp2 = tkp1 + Math.sqrt(Math.sqrt((pkp1.x-
      pkp2.x)**2 + (pkp1.y-pkp2.y)**2 + (pkp1.z-
      pkp2.z)**2 ))

  const samplesPerSegment = tkp1/deltat
  for (let j = 0; j < samplesPerSegment; j++) {
    const t = j*deltat;
    curvePoints.push(catmullRomcentripetal(pkm1,
        pk, pkp1, pkp2, tkm1,tkp1,tkp2,t));
  }
}
```

3.5 端点処理

Centripetal なパラメタ区分の場合においても、基本的には Uniform なパラメタ区分の場合と同様の端点処理をすることになる。すなわち、最初の制御点 $\mathbf{x}_0$ と $\mathbf{x}_1$ の間のパラメトリック曲線を描画する場合は $\mathbf{x}_{k-1} = \mathbf{x}_k = \mathbf{x}_0$ として描画するわけであるが、Centripetal な場合においてはこの時 $t_{k-1} = t_k = 0$ となってしまうことに注意しなければならない。というのも、$r_k(t)$ を逐次的に計算する際、l_{k-1} の計算で $t_k - t_{k-1}$ で割ると言う操作が出てきてしまうため $t_{k-1} = t_k$ となると都合が悪い。(下式参照)

$$l_{k-1}(t) = \left(1 - \frac{t - t_{k-1}}{t_k - t_{k-1}}\right)\mathbf{x}_{k-1} + \frac{t - t_{k-1}}{t_k - t_{k-1}}\mathbf{x}_k \tag{12}$$

そこで、$k = 0$ の場合は $l_{k-1}(t) = \mathbf{x}_{k-1}$ として直線 l_{k-1} を定数関数で表すことにする。(Figure8参照) こうすれば l_{k-1} と l_k の補間 q_k はこれまで通り二次曲線となり、以降のアルゴリズムは特別変更を受けない。(Figure9参照) [*9]

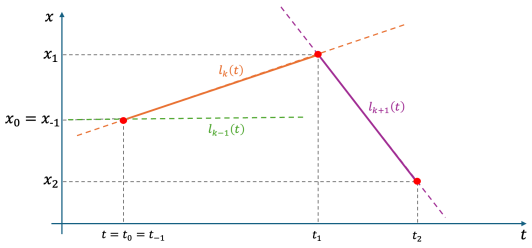

Figure8: Centripetal なパラメタ区分における端点処理。

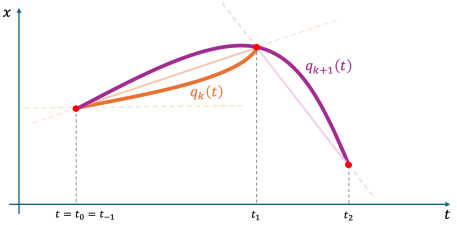

Figure9: 端点処理後の二次関数による補間。

[*8] $t_k = 0$ としていることに注意。

[*9] 最後の制御点 $\mathbf{x}_{N-1}$ と $\mathbf{x}_N$ の間のパラメトリック曲線の描画についても同様に処理する。

3.6　実装結果（Centripetal なパラメタ区分）

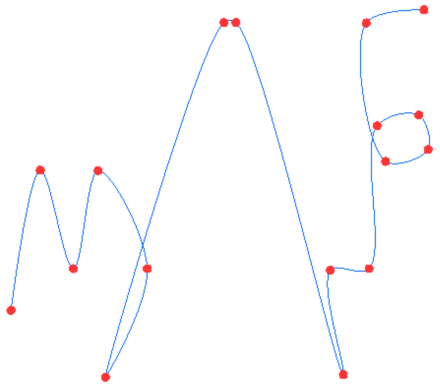

Figure10: Centripetal なパラメタ区分を用いた Catmull-Rom 曲線。

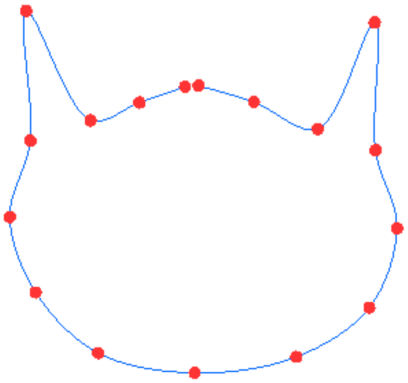

Figure11: Centripetal なパラメタ区分を用いた場合のネコ（?）の絵。

Centripetal なパラメタ区分を用いた場合の Catmull-Rom 曲線を描画した図が Figure10と Figure11である。まず Figure10をみると、大まかな挙動はパラメタがユニフォームな場合（Figure6）と変わっていないが、例えば図中央の制御点が密になっている箇所でループが生じていないなどの変化があることがわかる。すなわち、Centripetal なパラメタ区分を用いたことで望ましくない挙動を抑えることができている。

ただし、Figure10右側の制御点が四角形をなしている箇所や、左側のギザギザした場所を見ると、ユニフォームなパラメタ区分を用いた場合と同様「ぼわっと」曲線が膨らんでしまっていることがわかる。この点についてはのちの章で考察する。

また Figure11はネコの絵を Centripetal なパラメタ区分の Catmull-Rom 曲線で描いた図である。Figure7と比べると、ネコの「耳」の接続部に生じていた不自然な凹みがそれなりに解消されていることがわかる。ただし実際に絵を描いた感想としては、ユニフォームなパラメタ区分を用いた場合に比べて曲線が「ぼわっと」広がってしまうために望ましい曲線を得るのが困難だと感じた。

4　手法の比較

4.1　遠心力的な膨らみの抑制

Catmull-Rom 曲線においては、曲線の折り曲がりに際してその折れ曲がりとは逆方向へ曲線が膨らむ「遠心力」のような現象が見られることがわかった。

これを示したのが Figure12である。今ユニフォームなパラメタ区分で曲線を描いた Figure12左図を見ると、左から三番目の制御点の手前で曲線が折れ曲がりの向きとは逆方向に膨らんでしまっていることがわかる。このような現象はシャープな表現を困難にすると思われ、解消されることが望ましい。実際、Centripetal なパラメタ区分を用いた場合である Figure12右図を見ると、「膨らみ」が抑えられていることがわかり Centripetal なパラメタ区分の有用性が見て取れる。

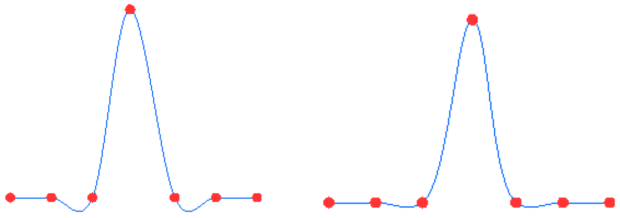

Figure12: 折れ曲がりに際して生じる折れ曲がりとは逆方向への曲線の膨らみ。左図：Uniform なパラメタ区分。右図：Centripetal なパラメタ区分

4.2　望ましくないループの抑制

次に、Centripetal なパラメタ区分によるループの抑制について考察する。ユニフォームなパラメタ区分において、制御点が密になっている箇所で望ましくないループが生じることはこれまで見てきた通りであり、それを示したのが Figure13左図である。（図中央は制御点が二つほぼ重なっている。）これは制御点間の距離が短いにも関わらず時間間隔 t が短くなっていないために無理やり曲線を描こうとしてしまうために生じる現象であり、Centripetal なパラメタ区分を用いることで抑制することができる。（Figure13右図参照。見にくいが、図中央は二つの制御点がほとんど重なっているにも関わらずループが生じていない。）

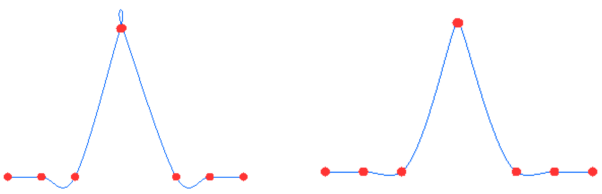

Figure13: 制御点が密に重なっている時の Catmull-Rom 曲線の挙動。左図：Uniform なパラメタ区分。右図：Centripetal なパラメタ区分

4.3　曲がり角が連続する領域

Figure14は制御点が四角形を成している時の Catmull-Rom 曲線を描いたものである。このような C^1 不連続な曲がり角が連続する場合は、Catmull-Rom 曲線の C^1 連続性が逆に仇となり曲線が「ぼわっと」膨らんでしまうことがわかる。また、これはパラメタの取り方の問題ではなく C^1 連続性という Catmull-Rom 曲線の根本設計によって生じる問題であるため、Centripetal なパラメタ区分を採用しても解消されない。(Figure14右図参照。)

そこで今回は、制御点を密にとることによりこの問題を解消することを試みた。これを表したのが Figure15である。この図においては、曲がり角付近で制御点を密にすることでシャープな折れ曲がりの連続を表現できるようになっている。

ただし Uniform なパラメタ区分においては、制御点間の距離が大きくなると曲がり角付近の制御点を密にしたことによってループの問題が出てきてしまう点に注意する必要がある。(Figure16参照。)

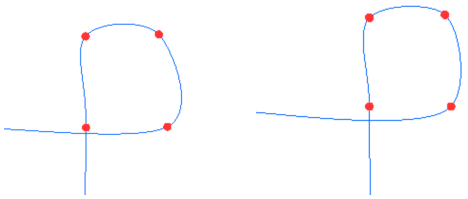

Figure14: 制御点が四角形を成している時の Catmull-Rom 曲線の挙動。左図：Uniform なパラメタ区分。右図：Centripetal なパラメタ区分

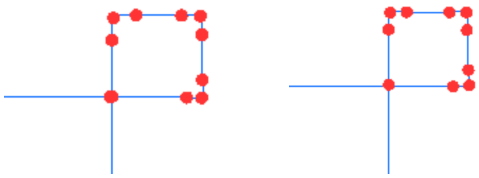

Figure15: 制御点を密にすることでシャープな曲がり角の連続を描くことができる様子。左図：Uniform なパラメタ区分。右図：Centripetal なパラメタ区分

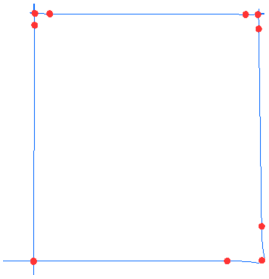

Figure16: 制御点間隔が大きい場合に、シャープな折れ曲がりの連続を折れ曲がり付近の制御点を密にすることで表現しようとした図。パラメタ区分は Uniform。この図を見ると、先に述べたループ現象の影響により折れ曲がりの点で Gibbs の角のようなものが現れてしまっている。また、右下の角は曲がり角付近の制御点間隔を少し広く取ることでループを抑えようとしているが、今度は逆にシャープな角の表現が難しくなってしまう。

5　おわりに

今回は Catmull-Rom 曲線を実装し、その挙動を確認した。その中で、Catmull-Rom 曲線が実際に滑らかな曲線となっていることや、制御点を必ず通るという制約により非常に直感的かつ簡単に絵を描くことができることなどがわかった。

また、Uniform なパラメタ区分と Centripetal なパラメタ区分の比較では、後者を用いることにより前者で現れる望ましくない挙動を抑制できることがわかった。ただし、Catmull-Rom 曲線の C^1 連続性に起因するシャープな曲がり角の描画の困難さは Centripetal なパラメタ区分でも解消できないこともわかった。

参考文献

[1] ja.Wikipedia Catmull-Rom スプライン曲線 https://ja.wikipedia.org/wiki/Catmull-Rom%E3%82%B9%E3%83%97%E3%83%A9%E3%82%A4%E3%83%B3%E6%9B%B2%E7%B7%9A