Dodecahedron 解説

吉川　亮

2021 年 9 月 14 日

目次

1 四元数と 3 次元の回転 3
1.1 四元数とは . . . 3
1.2 四元数の加法 . . . 3
1.3 四元数の実数倍 . . . 3
1.4 四元数の乗法 . . . 3
1.5 四元数と共軛，ノルム . . . 3
1.6 オイラーの公式の拡張 . . . 4
1.7 四元数と 3 次元の回転 . . . 4
1.8 ベクトルをベクトルに重ねる四元数 . . . 5

2 Ruby での四元数の実装：quaternion.rb 7
2.1 クラスとインスタンス変数 . . . 7
2.2 メソッド . . . 7
2.3 コンストラクタ . . . 7
2.4 演算子のオーバーライド . . . 8
2.5 実装した関数一覧 . . . 9
2.6 平方根の実装 . . . 9

3 立体の描画：draw3d.rb 10
3.1 Context3d クラスとインスタンス変数 . . . 10
3.2 Context3d クラスのメソッド . . . 10
3.3 面の色の計算 . . . 10
3.4 Draw3d モジュール . . . 10

4 正 12 面体データ：dodecahedron.rb 12
4.1 正 12 面体用 5 角形クラス FPentagon . . . 12
4.2 五角形の描画 . . . 12
4.3 正 12 面体クラス Dodecahedron . . . 13
4.4 正十二面体の頂点と面の計算 . . . 13

4.5 正十二面体の描画 15

5 メイン実行ファイル：main.rb 16
5.1 外部ソースを読み込む require と require_relative 16
5.2 glade ファイルの読み込み 16
5.3 グローバル変数 17
5.4 イベントハンドラ 17
5.5 DrawingArea への描画 18
5.6 ドラッグ動作の関連付け 19

付録 A 使用した Ruby 文法一覧 20
A.1 条件分岐 20
A.2 ループ 20
A.3 変数・定数・擬似変数 21
A.4 変数 21
A.5 関数・メソッド 22
A.6 外部ソースの読み込み 23

1　四元数と 3 次元の回転

1.1　四元数とは

複素数 $a+ib$ に対し，虚部を 3 次元に拡張した

$$a+ib+jc+kd \tag{1}$$

を四元数と言う．虚部の基底 i, j, k は以下の関係式を満たす．

$$i^2=j^2=k^2=-1 \tag{2}$$
$$ij=k, ji=-k \tag{3}$$
$$jk=i, kj=-i \tag{4}$$
$$ki=j, ik=-j \tag{5}$$

1.2　四元数の加法

四元数の加法は結合的で可換であり，以下のように定義される．

$$(a+ib+jc+kd)+(a'+b'i+c'j+d'k)=(a+a')+(b+b')i+(c+c')j+(d+d')k \tag{6}$$

1.3　四元数の実数倍

四元数の実数倍は各係数を全て実数倍する．

$$r\cdot(a+ib+jc+kd)=ra+rbi+rcj+rdk \tag{7}$$

1.4　四元数の乗法

四元数の乗法は (2)-(5) 式および (7) 式から計算できる．結合性ではあるが，非可換なことは留意．
特に純虚四元数どうしの積は，$q(x,y,z)=(i,j,k)\cdot(x,y,z)$ と表すことにすると

$$q(x,y,z)q(x',y',z')=-(xx'+yy'+zz')+q(yz'-y'z, zx'-z'x, xy'-x'y) \tag{8}$$
$$=-(x,y,z)\cdot(x',y',z')+q((x,y,z)\times(x',y',z')) \tag{9}$$

すなわち，実部が $(-1)\cdot$ 内積，虚部が外積になる．

1.5　四元数と共軛，ノルム

四元数 $q=a+ib+jc+kd$ の共軛を

$$\bar{q}=a-ib-jc-kd \tag{10}$$

で定義する．前項より

$$q\bar{q}=\bar{q}q=a^2+b^2+c^2+d^2 \tag{11}$$

となる．複素数の時と同様にここからノルム

$$\|q\| = \sqrt{q\bar{q}} = \sqrt{\bar{q}q} = \sqrt{a^2+b^2+c^2+d^2} \tag{12}$$

および逆数

$$q^{-1} = \bar{q}\cdot\|q\|^{-2} = (a-ib-jc-kd)(a^2+b^2+c^2+d^2)^{-1} \tag{13}$$

を定義できる．なお，四元数の乗法は非可換なため，除法での分数表記は避ける．左右の別が判然としないためである．

1.6 オイラーの公式の拡張

任意の正規純虚四元数 $\boldsymbol{v} = ib+jc+kd$ に対して

$$\boldsymbol{v}^2 = -b^2-c^2-d^2 = -1 \tag{14}$$

が成立する．従って，オイラーの公式を拡張して四元数 $r+\theta\boldsymbol{v}$(r および θ は実数) に対する指数函数を次のように定義できる．

$$\exp(r+\theta\boldsymbol{v}) = e^r(\cos\theta + \boldsymbol{v}\sin\theta) \tag{15}$$

ここから四元数 $q = a+\boldsymbol{v}b$ に対する対数函数の主値は

$$\mathrm{Log}(q) = \ln\|q\| + \boldsymbol{v}\mathrm{Arg}(q) \tag{16}$$

となり[*1]四元数 $q = a+\boldsymbol{v}b$ の実数 t 乗

$$q^t = \exp(t\mathrm{Log}(q)) \tag{17}$$

$$= \|q\|^t(\cos(t\mathrm{Arg}(q)) + \boldsymbol{v}\sin(t\mathrm{Arg}(q))) \tag{18}$$

を複素数同様に定義できる．[*2]特に，$a>0$ and/or $b\neq 0$ のとき

$$\sqrt{a+\boldsymbol{v}b} = \sqrt{\|a+\boldsymbol{v}b\|}\left(\cos\left(\frac{1}{2}\mathrm{Arg}(a+\boldsymbol{v}b)\right) + \boldsymbol{v}\sin\left(\frac{1}{2}\mathrm{Arg}(a+\boldsymbol{v}b)\right)\right) \tag{19}$$

$$= \sqrt[4]{a^2+b^2}\left(\sqrt{\frac{1}{2}+\frac{a}{2\sqrt{a^2+b^2}}} + \boldsymbol{v}\sqrt{\frac{1}{2}-\frac{a}{2\sqrt{a^2+b^2}}}\right) \tag{20}$$

$$= \sqrt{\frac{\sqrt{a^2+b^2}+a}{2}} + \boldsymbol{v}\sqrt{\frac{\sqrt{a^2+b^2}-a}{2}} \tag{21}$$

1.7 四元数と 3 次元の回転

以下断りなくベクトル (x,y,z) と純虚四元数 $(x,y,z)\cdot(i,j,k) = ix+jy+kz$ を同一視する．

$q = \cos\theta + \boldsymbol{v}\sin\theta$ ($\boldsymbol{v}$ は正規純虚四元数)、$\boldsymbol{u}$ も正規純虚四元数とすると，$\boldsymbol{v}$ を中心に $\boldsymbol{u}$ を 2θ だけ回転させたものは

$$q\boldsymbol{u}q^{-1} \tag{22}$$

[*1] ここで，$b=0$ の時も $a<0$ ならば右辺に $\boldsymbol{v}$ が登場することに注意．すなわち，四元数の枠組みでは負の実数の対数の主値を選ぶことができない．

[*2] ここでも負の実数の冪は実整数乗しか定義できないことに注意．

で表される．以下証明.

$$q^{-1} = \cos\theta - \boldsymbol{v}\sin\theta \tag{23}$$

から,

$$\begin{aligned} q\boldsymbol{u} &= \boldsymbol{u}\cos\theta + (-\boldsymbol{v}\cdot\boldsymbol{u} + \boldsymbol{v}\times\boldsymbol{u})\sin\theta && (24)\\ &= -(\boldsymbol{v}\cdot\boldsymbol{u})\sin\theta(\text{実部}) + u\cos\theta + (\boldsymbol{v}\times\boldsymbol{u})\sin\theta(\text{虚部}) && (25)\end{aligned}$$

$$\begin{aligned} q\boldsymbol{u}q^{-1} =& -(\boldsymbol{v}\cdot\boldsymbol{u})\sin\theta(\cos\theta - \boldsymbol{v}\sin\theta)\\ &+ (\boldsymbol{u}\cos\theta + (\boldsymbol{v}\times\boldsymbol{u})\sin\theta)\cos\theta\\ &- (\boldsymbol{u}\cos\theta + (\boldsymbol{v}\times\boldsymbol{u})\sin\theta)\cdot(-\boldsymbol{v}\sin\theta)\\ &+ (\boldsymbol{u}\cos\theta + (\boldsymbol{v}\times\boldsymbol{u})\sin\theta)\times(-\boldsymbol{v}\sin\theta) \end{aligned} \tag{26}$$

$$\begin{aligned} q\boldsymbol{u}q^{-1} =& -(\boldsymbol{v}\cdot\boldsymbol{u})\sin\theta\cos\theta + (\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{v}\sin^2\theta\\ &+ \boldsymbol{u}\cos^2\theta + (\boldsymbol{v}\times\boldsymbol{u})\sin\theta\cos\theta\\ &+ (\boldsymbol{u}\cdot\boldsymbol{v})\cos\theta\sin\theta + (\boldsymbol{v}\times\boldsymbol{u})\cdot\boldsymbol{v}\sin^2\theta\\ &- (\boldsymbol{u}\times\boldsymbol{v})\sin\theta\cos\theta + \boldsymbol{v}\times(\boldsymbol{v}\times\boldsymbol{u})\sin^2\theta \end{aligned} \tag{27}$$

実部は

$$\begin{aligned} Re(q\boldsymbol{u}q^{-1}) &= -(\boldsymbol{v}\cdot\boldsymbol{u})\sin\theta\cos\theta + (\boldsymbol{u}\cdot\boldsymbol{v})\cos\theta\sin\theta + (\boldsymbol{v}\times\boldsymbol{u})\cdot\boldsymbol{v}\sin^2\theta && (28)\\ &= 0 && (29)\end{aligned}$$

なので，これは純虚であり,

$$\begin{aligned} q\boldsymbol{u}q^{-1} =& (\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{v}\sin^2\theta\\ &+ \boldsymbol{u}\cos^2\theta + (\boldsymbol{v}\times\boldsymbol{u})\sin\theta\cos\theta\\ &- (\boldsymbol{u}\times\boldsymbol{v})\sin\theta\cos\theta + \boldsymbol{v}\times(\boldsymbol{v}\times\boldsymbol{u})\sin^2\theta \end{aligned} \tag{30}$$

ここで

$$\boldsymbol{u} - (\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{v} = \boldsymbol{v}\times(\boldsymbol{v}\times\boldsymbol{u}) \tag{31}$$

に注意して

$$\begin{aligned} quq^{-1} &= (\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{v}\sin^2\theta + (\boldsymbol{v}\times(\boldsymbol{v}\times\boldsymbol{u}) + (\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{v})\cos^2\theta + 2(\boldsymbol{v}\times\boldsymbol{u})\sin\theta\cos\theta + \boldsymbol{v}\times(\boldsymbol{v}\times\boldsymbol{u})\sin^2\theta && (32)\\ &= \boldsymbol{v}\times(\boldsymbol{v}\times\boldsymbol{u})\cos 2\theta + (\boldsymbol{v}\cdot\boldsymbol{u})\boldsymbol{v} + (\boldsymbol{v}\times\boldsymbol{u})\sin 2\theta && (33)\end{aligned}$$

これを (31) 式と比較すると，これは $\boldsymbol{v}$ を中心に $\boldsymbol{u}$ を 2θ だけ回転させたものであることがわかる

1.8 ベクトルをベクトルに重ねる四元数

$\boldsymbol{u},\boldsymbol{v}$ を正規純虚四元数とする．行列の時と同様に $\boldsymbol{u}$ を $\boldsymbol{v}$ に重ねる回転を表す四元数を求める．回転軸を $(\boldsymbol{u}\times\boldsymbol{v})\|\boldsymbol{u}\times\boldsymbol{v}\|^{-1}$ にとり，回転角を $\arccos(\boldsymbol{u}\cdot\boldsymbol{v})$ とすればよいので，求める四元数は

$$\sqrt{\frac{1+\boldsymbol{u}\cdot\boldsymbol{v}}{2}} + (\boldsymbol{u}\times\boldsymbol{v})\sqrt{\frac{1}{2(1+\boldsymbol{u}\cdot\boldsymbol{v})}} \tag{34}$$

となる．ところで，これを 2 乗すると

$$\left(\sqrt{\frac{1+\boldsymbol{u}\cdot\boldsymbol{v}}{2}}+(\boldsymbol{u}\times\boldsymbol{v})\sqrt{\frac{1}{2(1+\boldsymbol{u}\cdot\boldsymbol{v})}}\right)^2 = \frac{1+\boldsymbol{u}\cdot\boldsymbol{v}}{2}-\|\boldsymbol{u}\times\boldsymbol{v}\|^2\frac{1}{2(1+\boldsymbol{u}\cdot\boldsymbol{v})}+\boldsymbol{u}\times\boldsymbol{v} \tag{35}$$

$$= \frac{1+\boldsymbol{u}\cdot\boldsymbol{v}}{2}-\frac{1-(\boldsymbol{u}\cdot\boldsymbol{v})^2}{2(1+\boldsymbol{u}\cdot\boldsymbol{v})}+\boldsymbol{u}\times\boldsymbol{v} \tag{36}$$

$$= \boldsymbol{u}\cdot\boldsymbol{v}+\boldsymbol{u}\times\boldsymbol{v} \tag{37}$$

$$= -(-\boldsymbol{v})\cdot\boldsymbol{u}+(-\boldsymbol{v})\times\boldsymbol{u} \tag{38}$$

$$= -\boldsymbol{v}\boldsymbol{u} \tag{39}$$

したがって，求める四元数は以下のように単純な式で表せる．

$$\sqrt{-\boldsymbol{v}\boldsymbol{u}} \tag{40}$$

2　Ruby での四元数の実装：quaternion.rb

2.1　クラスとインスタンス変数

Ruby のクラス定義は class と end で囲まれたブロックで行う．Quaternion クラスでは基底 1, i, j, k の係数 r，i，j，k をインスタンス変数として格納している．attr_accessor は外部からインスタンス変数へのアクセスが可能になるメソッドを提供している．内部からインスタンス変数へのアクセスは@記号を用いる．

ソースコード 1　Quaternion クラス

```
1    class Quaternion # クラス宣言
2        # 外部からr，i，j，kへのアクセスを可能にする．
3        attr_accessor :r, :i, :j, :k
4        # ここにメソッドを実装する
5    end
```

2.2　メソッド

メソッドはクラス外の関数と同様に def で定義する．ここでは共軛を例にとる．new() は次項のコンストラクタを呼び出しインスタンスを作成する命令である．関数やメソッドの戻り値は次のように値のみの文を書く．(return を省略している．)

ソースコード 2　メソッド

```
1    class Quaternion
2        attr_accessor :r, :i, :j, :k
3        def conjugate() # メソッド
4            # アットマークを付けてインスタンス変数を呼び出す．
5            # return Quaternion.new(@r, -@i, -@j, -@k)
6            # のreturnを省略している
7            Quaternion.new(@r, -@i, -@j, -@k)
8        end
9    end
```

2.3　コンストラクタ

コンストラクタは initialize() 関数として定義する．Quaternion クラスでは，各インスタンス変数への代入のみ行っている．

ソースコード 3　Quaternion クラスのコンストラクタ

```
1    def initialize(r, i, j, k)
2        @r = r
3        @i = i
4        @j = j
5        @k = k
```

```
6       end
```

2.4 演算子のオーバーライド

Ruby では自作クラスに対して演算子を定義することができる．実装の方法は普通の関数と同様であり，ここでは 2 項演算である加算・減算・乗算および単項演算である負号を定義している．各 2 項演算の最初の行にある fail 文は演算の相手も四元数でないとエラーになるようにした命令であり，

ソースコード 4　fail 文

```
1       fail ’エラーメッセージ’
```

のように書く．Ruby では if と unless を命令の直後に書くことでも利用できる (修飾子という) が，通常の if 文または unless 文と動作は同じである．

ソースコード 5　if 文

```
1       if 評価式 then
2           命令文
3       end
```

ソースコード 6　if 修飾子

```
1       命令文 if 評価式
```

is_a?(class) はそのクラスのインスタンスかどうかを判定する．

ソースコード 7　二項演算子

```
1       def +(other) # 関数と同じ構文で演算子を定義．自身が左，otherが右である．
2           # otherが四元数でなければエラー．
3           raise TypeError.new(’Quaternion calc with non-quaternion’) unless other.is_a
                ?(Quaternion)
4           ar = @r + other.r
5           ai = @i + other.i
6           aj = @j + other.j
7           ak = @k + other.k
8           Quaternion.new(ar, ai, aj, ak)
9       end
```

単項演算子には + および-の 2 種類があり，それぞれ +@，-@という名前で定義することでオーバーライドできる．

ソースコード 8　単項演算子

```
1       def -@
2           Quaternion.new(-@r, -@i, -@j, -@k)
3       end
```

2.5 実装した関数一覧

四元数の実装を完全に行うならば各種超越関数なども用意するべきではあるが，今回は簡易的なプログラムのため，以下の関数および演算子のみ定義した．なお，特に断りのないものは四元数を返す．

- コンストラクタ (initialize)
- 加算 (+)
- 減算 (-)
- 乗算 (*)
- 負号 (-@)
- ノルムの 2 乗 (norm_square)，実数
- 共軛 (conjugate)
- 逆数 (inv)
- 虚部 (im)
- ノルム，または絶対値 (abs)，実数
- 正規化 (normalize)
- 平方根 (sqrt)

2.6 平方根の実装

平方根は (1.6) 節の通り実装すればよいが，ここでは次の通り変形する．

$$\sqrt{a+\boldsymbol{v}b} = \sqrt{\frac{\sqrt{a^2+b^2}+a}{2}} + \boldsymbol{v}\sqrt{\frac{\sqrt{a^2+b^2}-a}{2}} \tag{41}$$

$$= \sqrt{\frac{\sqrt{a^2+b^2}+a}{2}} + \boldsymbol{v}\sqrt{\frac{b^2}{2(\sqrt{a^2+b^2}+a)}} \tag{42}$$

$$= \sqrt{\frac{\sqrt{a^2+b^2}+a}{2}} + \frac{\boldsymbol{v}b}{2}\sqrt{\frac{2}{\sqrt{a^2+b^2}+a}} \tag{43}$$

実装では i, j, k それぞれの係数を保持しており，正規ベクトル $\boldsymbol{v}$ と係数 b に分けて管理しているわけではないので，こちらの方が簡潔なコードになる．

ソースコード 9　四元数の平方根の実装

```
1    def sqrt
2        r_sqrt = Math::sqrt((self.abs()+@r)/2.0)
3        Quaternion.new(r_sqrt, @i/2.0/r_sqrt, @j/2.0/r_sqrt, @k/2.0/r_sqrt)
4    end
```

3　立体の描画：draw3d.rb

3.1　Context3d クラスとインスタンス変数

Context3d クラスは 3 次元オブジェクトを描画するためのクラスであり，インスタンス変数として以下を定義している．

- context: 2 次元オブジェクトを描画するためのクラスのインスタンス．
- q: 回転四元数．
- qlight: 光源の方向を表す純虚四元数．
- r，g，b: 光源の色．コンストラクタの時点では白色．

3.2　Context3d クラスのメソッド

Context3d クラスのメソッドとして以下を実装している．

- initialize(context，q，qlight): コンストラクタ．引数の=以下は，引数を省略した時のデフォルトを表している．
- set_light_color(r，g，b): 光源の色を設定する．
- clear(r，g，b):指定した色でキャンバスを塗りつぶす．
- move_to(qp): qp は位置を表す四元数であり，線描画の始点をここに移動する．qp を@q で回転させた座標を二次元に圧縮し，2 次元キャンバス@context の move_to メソッドを呼び出している．
- line_to(qp): 現在のペンの位置から qp まで線を引く．実装は move_to(qp) とほぼ同じ．
- set_brightness(q): 引数 q は面の法線ベクトルを表す四元数．光源の角度と面の角度から明るさを計算し，線や面の色として@context に保存しておく．

3.3　面の色の計算

面の色として，光源の色を (r, g, b), 回転四元数を q, 面の法線ベクトルを表す四元数を ε, 光源の位置を表す四元数を λ とすると，面の色を次の式によって決定している．

$$\frac{1}{2} + \frac{1}{2}\mathrm{Re}\left(q\varepsilon q^{-1}\lambda\right) \tag{44}$$

Re の部分は回転後面の法線ベクトル $(q\varepsilon q^{-1})$ と光源ベクトル (λ) の内積の -1 倍である．

3.4　Draw3d モジュール

このモジュールは実数にしたマウス座標 (x, y) を空間座標を表す純虚四元数に変換するメソッドを提供している．z 座標の決定に当たっては，画面中央に半球の存在を仮定し，マウスポインタが半球の中にある時は半球の表面までせり出させている．具体的な z 座標は

$$z = \begin{cases} 0 & (x^2 + y^2 > r^2) \\ \sqrt{\rho^2 - (x^2 + y^2)} & otherwise \end{cases} \tag{45}$$

となる．実装では $\rho = 1$ としている．モジュール内のメソッドはそのままでは外部から呼び出せないため，以下のようにモジュール関数として定義している．

ソースコード 10　モジュール関数

```
module Draw3d
    def p_to_q(x, y)
        r = x*x+y*y
        if(r > 1.0)
            r = Math::sqrt(r)
            x /= r
            y /= r
            z = 0.0
        else
            z = Math::sqrt(1.0-r*r)
        end
        Quaternion.new(0, x, y, z) # 戻り値
    end
    module_function :p_to_q # モジュール関数としてexportするための命令
end
```

4　正 12 面体データ：dodecahedron.rb

4.1　正 12 面体用 5 角形クラス FPentagon

このクラスは 5 角形を描画するクラスであり，インスタンス変数 points に頂点を表す 5 つの四元数からなる配列を格納する．メソッドは以下の通り．

- initialize(points): コンストラクタ．
- draw(context3d): 3 次元キャンバス context3d に 5 角形を描画する．
- center: 5 角形の重心を求める．

draw および center メソッドにある for 文は，配列内のすべての要素に対してループを回す所謂 foreach の役割を果たす．

ソースコード 11　for 文

```
1     def center
2         aq = Quaternion.new(0, 0, 0, 0)
3         z = Quaternion.new(0.2, 0, 0, 0)
4         for q in @points do # @points内の全ての要素qに対してループ
5             aq = aq + q
6         end
7         aq * z # 戻り値
8     end
```

4.2　五角形の描画

FPentagon クラスの draw メソッドは Context3d クラスを引数に取り，そこに自身を描画するメソッドである．

```
1     def draw(context3d)
2         # 面の色(明るさと光源色の積)を計算
3         # 原点を中心とした正二十面体の面のため，
4         # 法線ベクトルと重心の位置ベクトルが平行になる．
5         context3d.set_brightness(-self.center)
6         # あらかじめpointsの末尾にmove_toしておく．
7         context3d.move_to(@points[4])
8         # 辺を描画
9         for q in @points do
10            context3d.line_to(q)
11        end
12        # Context3dクラスのfillメソッドを用いて描画した面を塗りつぶす．
13        context3d.context.fill
14    end
```

4.3 正 12 面体クラス Dodecahedron

このクラスは原点を中心とする正 12 面体を描画するクラスである．インスタンス変数 l は一辺の長さであり，faces は正十二面体を構成する FPentagon クラスの配列である．Phi は黄金比であり，Dodecahedron クラスの末尾にある private_constant でクラス内のプライベート定数として振る舞うようにしている．なお，変数は小文字，定数は大文字で始まる識別子を使う．

ソースコード 12　Dodecahedron クラスのインスタンス変数とプライベート定数

```
1  class Dodecahedron
2      # インスタンス変数
3      attr_accessor :l, :faces
4
5      Phi = (1.0+Math.sqrt(5.0))/2.0
6
7      # メソッド
8
9      # プライベート定数
10     private_constant :Phi
11 end
```

4.4 正十二面体の頂点と面の計算

Dodecahedron クラスではコンストラクタで座標の計算を行っている．コンストラクタでは引数に一辺の長さ l を取り，ここから@faces に格納する座標・面データを計算している．一辺 1 の正十二面体の頂点のうち，8 頂点を xyz 軸に沿う立方体とした時の頂点の座標は以下の通りである．ここではすべて複合任意である．ϕ を黄金比とする．

$$\left(\pm\frac{\phi}{2},\pm\frac{\phi}{2},\pm\frac{\phi}{2}\right)$$
$$\left(0,\pm\frac{1}{2},\pm\frac{1+\phi}{2}\right)$$
$$\left(\pm\frac{1}{2},\pm\frac{1+\phi}{2},0\right)$$
$$\left(\pm\frac{1+\phi}{2},0,\pm\frac{1}{2}\right)$$

この時，同じ面に属する点を辺に沿ってなぞると，例えば以下のようになる．

$$\left(-\frac{1}{2},\frac{1+\phi}{2},0\right)\to\left(-\frac{\phi}{2},\frac{\phi}{2},\frac{\phi}{2}\right)\to\left(0,\frac{1}{2},\frac{1+\phi}{2}\right)\to\left(\frac{\phi}{2},\frac{\phi}{2},\frac{\phi}{2}\right)\to\left(\frac{1}{2},\frac{1+\phi}{2},0\right)$$

この各頂点を (ξ,η,ζ) とすると，$(\xi,\pm\eta,\pm\zeta)$ の配列，$(\pm\eta,\pm\zeta,\xi)$ の配列，$(\pm\zeta,\xi,\pm\eta)$ の配列 (すべて複合任意) を計算することで全 12 面を構成できる．なお，ソースコード中の 12.times do |counter|および array.each do |element|は for counter in 0..11 や for element in array とほぼ同じであるが，ループ変数が for 文ではループ外からもアクセスできるのに対して do ループでは内部からしかアクセスできないという違いがある．

ソースコード 13　Dodecahedron クラスのコンストラクタ

```
1   def initialize(l)
2       @l = l
3       # 空の配列を作成．面を表すFPentagonクラスの配列になる．
4       @faces = Array.new()
5
6       # 各面の頂点を計算するためのテンプレート．
7       bpoints = Array.new()
8       bpoints = bpoints + [[-1.0, 1.0+Phi, 0]]
9       bpoints = bpoints + [[-Phi, Phi, Phi]]
10      bpoints = bpoints + [[0, 1.0, 1.0+Phi]]
11      bpoints = bpoints + [[Phi, Phi, Phi]]
12      bpoints = bpoints + [[1.0, 1.0+Phi, 0]]
13
14      # timesメソッドを用いたループ．
15      12.times do |counter|
16          # counterが偶数のとき-0.5，奇数のとき0.5
17          x = counter%2 - 0.5
18          # counterが0,1 mod 4のとき-0.5，2,3 mod 4のとき0.5．
                xと併せて複合任意のため．
19          y = counter/2%2 - 0.5
20          # 頂点のxyzの順番を回すための変数．
21          # counterが0..3のときxyzのまま，4..7のときzxyの順番，8..11のとき
                yzxの順番．
22          order = counter/4
23          # 面を構成する1頂点の座標．長さ3の実数の配列．各要素の初期値はnil
24          fpoint = Array.new(3)
25          # 面を構成する全頂点に対応する四元数の配列．
26          # FPentagonクラスのコンストラクタに渡す．
27          qs = Array.new()
28          # eachメソッドを用いたループ．
29          bpoints.each do |point|
30              fpoint[order%3] = point[0]/2.0*l
31              fpoint[(order+1)%3] = point[1]*x*l
32              fpoint[(order+2)%3] = point[2]*y*l
33              # 純虚四元数に変換して配列qsの末尾に追加．
34              # Quaternionの外の[](角括弧)は配列を表す．
35              # つまり正確には配列qsと長さ1の配列をドッキングしている．
36              qs = qs + [Quaternion.new(0.0, fpoint[0], fpoint[1], fpoint[2])]
37          end
38          # qsをFPentagonクラスのコンストラクタに渡して，
39          # インスタンスを@facesの末尾に追加．
40          @faces = @faces + [FPentagon.new(qs)]
41      end
42  end
```

4.5 正十二面体の描画

Dodecahedron クラスの draw メソッドは Context3d クラスを引数にとり，これに自身を描画するメソッドである．このメソッドでは隠面処理として最も単純な画家のアルゴリズムを採用している．すなわち，各面を z 座標でソートし，奥から手前に順に描画していく．正十二面体のような単純な図形で使用可能なアルゴリズムであるが，複雑な図形の場合はより複雑なアルゴリズムを採用する必要がある．Array クラスの sort_by メソッドを利用することで，式を用いた比較ができる．

`array.sort_by{|element| key(element)}`

と書くと array の要素ごとに key を計算し，昇順にソートする．ここでは

- 各面を表す Pentagon クラスのインスタンス face ごとに，
- その重心を q で回転させた後の z 座標 `q*face.center*(q.inv).k` を計算し，
- z 座標昇順にソート

という処理が行われる．

sort_by は新たな配列を返すメソッドであり，ここではその返ってきた配列に対して each メソッドでループをしている．中括弧 {} が続いているが，仕様は前述の do を使用したループと同じである．

ソースコード 14　正十二面体の描画

```
1     def draw(context3d)
2         # 回転四元数
3         q = context3d.q
4         # ここの解説は本文参照
5         @faces.sort_by {|face| (q*face.center*(q.inv)).k }.each {|face| face.draw(
              context3d)}
6     end
```

5　メイン実行ファイル：main.rb

5.1　外部ソースを読み込む require と require_relative

require および require_relative は外部のソースやライブラリを読み込む命令である．Ruby の標準ライブラリや gem install したライブラリを読み込むには単に require 'library' とすればよい．自作のソースを読み込む際は require はカレントディレクトリ (コマンドラインで ruby を実行したディレクトリ) からの相対パスとして解釈するため，カレントディレクトリが違うと実行できなくなる．そこで，require_relative を使う．こちらはソースが存在するディレクトリからの相対パスとして解釈するため，環境に依存せずに実行できる．

ソースコード 15　外部ソース・ライブラリの読み込み

```
1  # gem installしたgtk3ライブラリはrequireで読み込める
2  require 'gtk3'
3
4  # main.rbからの相対パスで読み込みたいのでrequire_relativeを使う
5  require_relative './utils/draw3d'
6  require_relative './utils/quaternion'
7  require_relative './utils/dodecahedron'
```

5.2　glade ファイルの読み込み

このアプリケーションではウインドウの情報を xml 形式の glade ファイルから読み込んでいる．オブジェクトの情報はこの glade ファイルに入っており，イベントハンドラ名などもここで指定してある．glade ファイルは gtk3 をインストールすると付いてくる Glade というアプリを使って GUI で作成できる (ここを参照のこと) ため，その内容や文法については割愛する．glade ファイルの読み込みには Gtk::Builder を使い，各オブジェクトの取得にはこれの get_object() メソッドを用いる．

ソースコード 16　object の読み込み

```
1  # File.dirname(__FILE__)はmain.rbのあるディレクトリの絶対パス．
2  # File.join()はファイルのURLをつなげる．
3  builder = Gtk::Builder.new(file: File.join(File.dirname(__FILE__), 'test.glade'))
4
5  # get_objectの引数はGladeで指定した名前と一致させる．
6  window = builder.get_object('Window0')
7  drawingarea = builder.get_object('DrawingArea0')
8
9  # 初期化とイベントハンドラをここに
10
11 # windowを表示
12 window.show_all
13 # Gtkのメインループを実行
14 Gtk.main
```

5.3 グローバル変数

$で始まる変数はグローバル変数であり，どこからでも参照できる．ここではマウス座標のバッファ，ドラッグ開始フラグ，回転四元数，正十二面体クラスのインスタンスおよび描画用の Context3d クラスのインスタンスをグローバル変数に格納している．

ソースコード 17　グローバル変数

```
1  # マウス座標のバッファ
2  $mousex = 0
3  $mousey = 0
4  # ドラッグ開始フラグ
5  $dragjustbegun = false
6  # 回転四元数
7  $rotateq = Quaternion.new(1.0, 0.0, 0.0, 0.0)
8  # 正十二面体
9  $dodecahedron = Dodecahedron.new(120.0)
10 # 三次元描画コンテキスト
11 $context3d = Context3d.new
```

5.4 イベントハンドラ

イベントハンドラは Glade で指定した名前の関数を実装し，connect_signals メソッドで関連付けを行う．例えば，ウインドウが破棄された時の処理部分を抜粋すると以下のようになる．

ソースコード 18　イベントハンドラ

```
1  def on_win_destroy
2      # プログラムを終了
3      Gtk.main_quit
4  end
5
6  # handlerにはgladeファイルで定義されたイベントハンドラ名が格納される．
7  # method(handler)は文字列であるhandlerから対応するmain.rb内の関数を呼び出す．
8  builder.connect_signals{ |handler| method(handler)}
```

なお，今回はあらかじめ on_win_destroy と window.destroy が関連付けられた glade ファイルを用意したが，そうせずに Ruby のみで完結させる方法もある．

ソースコード 19　Ruby コード内でのイベントハンドラの実装

```
1  # windowを作成
2  window = Gtk::Window.new
3  # windowのdestroyシグナルとon_win_destroy関数を関連付け
4  # gladeファイルにsignal文が無い場合，
5  # connect_signalsで一気に関連付けをすることはできない
6  window.signal_connect('destroy'){on_win_destroy}
```

5.5 DrawingArea への描画

このプログラムでは DrawingArea の draw イベントハンドラ，dragbegin イベントハンドラおよび drag-motion イベントハンドラを用いて描画を行っている．on_drawingarea_draw は draw シグナルに関連付けられていて，DrawingArea に描画・再描画が行われる時に呼び出される．

ソースコード 20　draw メソッド

```
1  # 引数widgetにはdrawingarea(Gtk::DrawingAreaクラス)が，
2  # contextにはCairo::Contextクラスのインスタンスが格納される．
3  def on_drawingarea_draw(widget, context)
4      # 原点をdrawingareaの中央に移動(デフォルトでは左上)
5      context.translate(widget.allocated_width/2.0, widget.allocated_height/2.0)
6
7      # draw3d.rbのContext3dクラスのインスタンスに渡す
8      $context3d.context = context
9      # 白で塗りつぶす
10     $context3d.clear(1.0, 1.0, 1.0)
11     # 正十二面体を描画
12     $dodecahedron.draw($context3d)
13
14     # 二次元Contextを破棄
15     # 明示的にdestroyしないとプログラム終了時エラーになる
16     context.destroy
17 end
```

on_drawingarea_dragbegin は drag-begin シグナルに関連付けられており，描画開始のフラグを立てている．on_drawingarea_dragmotion は drag-motion シグナルに関連付けられており，ドラッグ中の処理を行っている．具体的にはマウスの動きから回転四元数の変化を計算し，DrawingArea の再描画を enqueue している．

ソースコード 21　ドラッグ処理

```
1  def on_drawingarea_dragbegin(widget, context)
2      $dragjustbegun = true
3  end
4
5  def on_drawingarea_dragmotion(widget, context, x, y, time)
6      # ドラッグ開始直後はバッファにマウス座標を格納して終わり
7      if $dragjustbegun then
8          $mousex = x
9          $mousey = y
10         $dragjustbegun = false
11         # return文が実行されるとそれ以降の文は実行されない
12         return
13     end
14
```

```
15      # マウス座標を正規純虚四元数に変換
16      # 描画の原点は中央になったが，マウス座標の原点は左上のままなので
17      # 中央に持ってくる
18      fromi = ($mousex - widget.allocated_width/2.0)/300.0
19      fromj = ($mousey - widget.allocated_height/2.0)/300.0
20      fromq = Draw3d::p_to_q(fromi, fromj)
21      toi = (x - widget.allocated_width/2.0)/300.0
22      toj = (y - widget.allocated_height/2.0)/300.0
23      toq = Draw3d::p_to_q(toi, toj)
24
25      # 回転四元数を更新
26      $rotateq = (-toq*fromq).sqrt*$rotateq
27      $context3d.q = $rotateq
28      # 再描画をenqueueする
29      widget.queue_draw
30
31      # バッファに現在のマウス座標を格納
32      $mousex = x
33      $mousey = y
34  end
```

5.6 ドラッグ動作の関連付け

ドラッグ動作の実装にはあと一手間が必要である．Gtk::TargetEntry クラスを用いてドラッグの有効範囲，ドラッグに使用するマウスボタンなどを設定する．ここに関しては作者もいまいち分かっていないので，詳しくはGtk::TargetEntry(リンク) およびGtk::Widget(リンク) を参照のこと．

ソースコード 22　ドラッグの設定

```
1  targetentry = Gtk::TargetEntry.new("DrawingArea0", Gtk::TargetFlags::SAME_WIDGET, 0)
2  drawingarea.drag_source_set(Gdk::ModifierType::BUTTON1_MASK, [targetentry], Gdk::
       DragAction::PRIVATE)
3  drawingarea.drag_dest_set(Gtk::DestDefaults::ALL, [targetentry], Gdk::DragAction::
       PRIVATE)
```

付録 A　使用した Ruby 文法一覧

A.1　条件分岐

A.1.1　if 文，if 修飾子

```
1  # 式がtrueならば
2  if expression then
3      # 処理
4
5  # そうでなくて，次の式がtrueならば
6  elsif expression then
7      # 処理
8
9  # どれもtrueでなければ
10 else
11     # 処理
12 end
13
14 # 修飾子
15 instruction if expression
```

A.1.2　unless 文，unless 修飾子

```
1  # 式がfalseならば
2  # elsifおよびelseは使えない
3  unless expression then
4      # 処理
5  end
6
7  # 修飾子
8  instruction unless expression
```

A.2　ループ

A.2.1　each メソッド

```
1  # 配列array内のすべてのelementに対して
2  array.each{ |element|
3      # 処理
4  }
5  # doを用いた書き方
6  array.each do |element|
7      # 処理
8  end
```

A.2.2 for ループ

```
# 配列array内のすべてのelementに対して
for element in array do
    # 処理
end
```

A.2.3 times メソッド

```
# 0から11まで順に
12.times { |counter|
    # 処理
}
# doを用いた書き方
12.times do |counter|
    # 処理
end
```

A.3 変数・定数・擬似変数

A.4 変数

```
# ローカル変数
x = 0
# グローバル変数
$y = 0
# インスタンス変数
@z = 0
```

A.4.1 擬似変数

```
# メソッドの実行主体
self
# いわゆるnull
nil
# 真
true
# 偽
false
# ソースファイル名
__FILE__
```

A.4.2 定数

```
# 大文字で始める
X = 0
```

A.4.3 クラス・モジュール外からのアクセス

```
class ClassA
    attr_accessor :a # 外部からアクセス可能にする
    def initialize
        @a = 0
        @b = 1
        A = 2
    end
end

instanceA = ClassA.new
# インスタンスの変数・関数へのアクセスにはドットを使う
print instanceA.a # 0
print instanceA.b # エラー
# モジュールやクラスの定数へはダブルコロンを使う
print ClassA::A # 2
```

A.5 関数・メソッド

A.5.1 定義

```
def function_name(arg1, arg2)
    # 処理

    # 返り値，returnは省略可
    return res
end
```

A.5.2 モジュール関数

```
module ModuleA
    def function_a
        # 処理
    end
    # モジュール関数にする
    module_function :function_a
end

```

```
9  # 呼び出し
10 ModuleA.function_a
```

A.6 外部ソースの読み込み

```
1  # 標準ライブラリ，gem installしたライブラリ
2  require 'gtk3'
3
4  # 自作ライブラリ
5  # ソースからの相対パス
6  # 拡張子は省略できる
7  require_relative 'lib/mylibrary'
```