

コンピュータグラフィックス特論 II

第7回 キーフレームアニメーション(2)

九州工業大学 尾下 真樹

2021年度

0

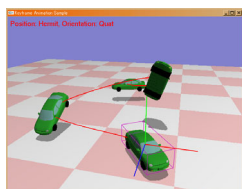
今日の内容

- 向きの補間
 - オイラー角
 - 四元数と球面線形補間
 - 相互変換
- アニメーションプログラミング
- レポート課題

1

キーフレームアニメーション

- 入力された複数のキーフレーム(時刻・状態の組)からアニメーションを生成
 - 少数のキーフレームの情報から、連続的なアニメーションを生成
 - 前後のキーフレームの状態(位置・向き)を補間して、キーフレーム間の任意時刻の状態を生成
 - 位置や向きの補間の計算が必要となる



2

全体の内容

- キーフレームアニメーションの基礎
- サンプルプログラム
- 行列・ベクトルを扱うプログラミング
- 位置補間
 - 線形補間、Hermite曲線、Bézier曲線、B-Spline曲線
- 向きの補間
 - オイラー角、四元数と球面線形補間、相互変換
- アニメーションプログラミング
- レポート課題

3

今日の内容

- 向きの補間
 - オイラー角
 - 四元数と球面線形補間
 - 相互変換
- アニメーションプログラミング
- レポート課題

4

参考書

- 「3DCGアニメーション」
栗原恒弥・安生健一 著、技術評論社、¥2,980
 - アニメーション技術全般を解説
- 3次元図形処理工学
黒瀬 能幸 著、共立出版、¥2,600
 - 曲線・曲面について詳しく説明
- vecmathを理解するための数学
平鍋 健児 著(四元数の詳しい解説)
 - <http://www.objectclub.jp/download/vecmath1>



5

参考書(続き)

- Computer Graphics Gems JP 2013/2014
「パラメトリックポーズブレンド」

– 回転の補間方法についての詳しい解説

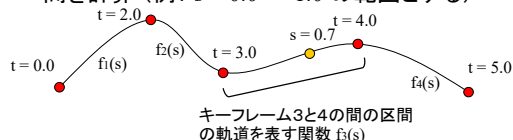


6

補間の考え方(復習)

• 補間関数

- 軌道全体を各キーフレーム間の区間に分ける
- 各区間の軌道を何らかの関数により表現
 - 通常は、区間の前後の制御点をもとに、関数を決定
- 全体の時刻から、現在の区間内のローカル時間を計算 (例: $s = 0.0 \sim 1.0$ の範囲とする)



7

位置・向き(復習)

• 位置の補間方法

- 位置の表現方法
 - 位置ベクトルによる表現
- 位置の補間方法
 - 線形補間、Hermite曲線、Bézier曲線、B-Spline曲線

• 向きの補間方法

- 向きの表現方法
 - 回転行列、オイラー角、回転軸と回転角度、四元数
- 向きの補間方法
 - オイラー角、四元数

8



9

向きの補間

10

向きの表現と補間方法

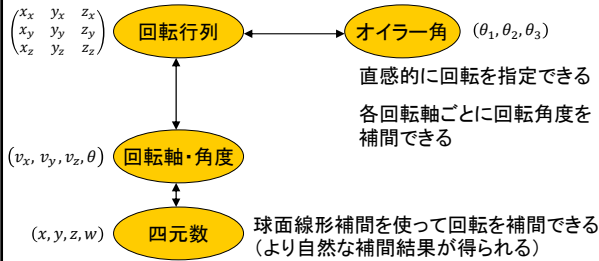
• 向きの表現方法

- 回転行列による表現 (3×3 行列) $\begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$
 - 基本的な表現方法 (OpenGL等に向きの情報を渡すときには、回転行列で表現する必要がある)
 - 余計なデータが多い、補間は難しい
- オイラー角による表現 $(\theta_1, \theta_2, \theta_3)$
 - 各回転軸ごとに回転角度を補間できる
- 回転軸と回転角度による表現 (v_x, v_y, v_z, θ)
- 四元数による表現 (x, y, z, w)
 - 球面線形補間を使って向きを全体的に補間できる

11

向きの表現方法と相互変換

- 回転行列による表現方法が基本



12

向きと回転の関係

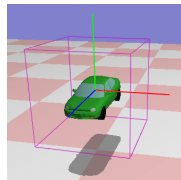
- 向きと回転の違いは何か？
- 向きは回転によって表現できる
 - 初期状態からの回転により表現
 - 一つの向きを複数の回転により表せる
 - 例: Y軸周りに90度回転、-270度回転、450度回転は、全て同じ向きになる
 - 表現を一通りにするためには、何らかの制約が必要
- 回転は向きでは表せない
 - 180度を超える回転は、向きでは表せない

13

回転行列による表現

- 回転行列 (3×3行列) による表現
 - 各列が、ワールド座標系における、モデル座標系のX軸・Y軸・Z軸の方向ベクトルを表す
 - 各列の長さは1で、互いに直交する必要がある
 - 向きが一意に決まる
 - 一つの向きの表現方法は、一通りしかない

$$\mathbf{M} = \begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$$



14

回転行列の補間

- そのまま補間することは難しい
 - 3×3行列の各要素を別々に補間すると、回転行列の制約が満たされない
 - 各列の長さは1で、互いに直交する必要がある

$$\mathbf{M} = \begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$$

15

オイラー角による表現

- 各軸周りの回転角度 (Θ) の組で向きを表現
 - 回転行列の積によって全体の向きを計算

$$\mathbf{M} = R_z(\theta_2) \cdot R_x(\theta_1) \cdot R_y(\theta_0)$$

$$= \begin{pmatrix} \cos \theta_2 & -\sin \theta_2 & 0 \\ \sin \theta_2 & \cos \theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 \\ 0 & \sin \theta_1 & \cos \theta_1 \end{pmatrix} \begin{pmatrix} \cos \theta_0 & 0 & \sin \theta_0 \\ 0 & 1 & 0 \\ -\sin \theta_0 & 0 & \cos \theta_0 \end{pmatrix}$$

- ※ 回転行列の適用順序によって向きが変わる
 - 適切な軸と順序をあらかじめ決めておく必要がある
 - 方位角 (y軸周りの回転) → 仰角 (x軸周りの回転) → 回転角 (z軸周りの回転) がよく使われる

16

回転行列からオイラー角への変換

- オイラー角表現での、回転軸の適用順序によって異なる
- 前スライドの行列 ($\Theta_0 \sim \Theta_2$ の式) の、回転行列の各要素 (3×3) の連立方程式より計算
- そのままでは複数の解が存在してしまうので、回転角度の範囲に関する仮定を置く必要がある
 - 例: 仰角は -90度~90度の範囲 など

17

回転行列からオイラー角への変換

• 例: 方位角 → 仰角 → 回転角 の場合

- 仰角は -90度~90度の範囲と仮定
- Z軸の x座標・z座標から、方位角を計算

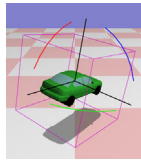
$$\theta_{yaw} = \tan^{-1} \frac{Z_x}{Z_z} \quad \text{or} \quad \tan^{-1} \frac{Z_x}{Z_z} + \pi (Z_z < 0)$$

- Z軸の y座標・xz座標から、仰角

$$\theta_{pitch} = \tan^{-1} \frac{\cos \theta_{yaw} Z_y}{Z_z} \quad \left(\text{or} \quad \tan^{-1} \frac{Z_y}{\sqrt{Z_x^2 + Z_z^2}} \right)$$

- Y軸の回転から、回転角を計算

$$\theta_{roll} = \tan^{-1} \frac{\cos \theta_{pitch} (\sin \theta_{yaw} Y_z - \cos \theta_{yaw} Y_x)}{Y_y}$$



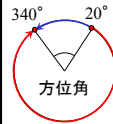
18

オイラー角の補間

• 各回転角度 ($\Theta_0 \sim \Theta_2$) を独立に補間

- 位置の補間方法と同じ方法がそのまま適用可能
- 方位角の補間には、注意が必要

- 仰角は -90~90度の間、回転角は -180~180度の間で変化すると仮定できる(これらの範囲を超えない)
- 方位角は、0~360度 (or -180~180度など) の間で変化するか、範囲の境界は連続している



- 例: 20度 → 340度に変化するときには、+320度ではなく、-40度の方向に変化するべき
- 2つの向きの差が 180度以下になるように変換してから補間
- 例: 20度 → 380度に変換してから、380度と340度の間を補間

19

オイラー角の補間の問題

• 2つの方向の間が真っすぐに補間されない

- オイラー角による表現では、前の回転軸周りの回転により、次の回転軸が回転して影響を受けるため

- 四元数による表現を使うことで、問題を解決できる

20

プログラム例(1)

• オイラー角による向きの補間の処理の流れ

```
// 区間の両端点の向きを取得
const Matrix3f & o0 = keyframes[ seg_no ].ori;
const Matrix3f & o1 = keyframes[ seg_no + 1 ].ori;

// オイラー角表現に変換
float y0, p0, r0;
float y1, p1, r1;
ConvMatToEular( o0, y0, p0, r0 );
ConvMatToEular( o1, y1, p1, r1 );

// 方位角(y0, y1)、仰角(p0, p1)、回転角(r0, r1)
// 回転行列からオイラー角表現への変換
// 前のスライドの計算を実装した関数
...

// 回転行列からオイラー角への変換 (yaw → pitch → rollの順の場合)
void ConvMatToEular( const Matrix3f & m, float & yaw, float & pitch, float & roll )
{
    ...
}
```

21

プログラム例(2)

• オイラー角による向きの補間の処理の流れ

```
// 各回転角度を線形補間
float y, p, r;
if ( y0 < y1 - M_PI )
    y0 += 2.0f * M_PI;
else if ( y0 > y1 + M_PI )
    y0 -= 2.0f * M_PI;

y = ( y1 - y0 ) * t + y0;
p = ( p1 - p0 ) * t + p0;
r = ( r1 - r0 ) * t + r0;

// 行列表現に変換して出力
...
```

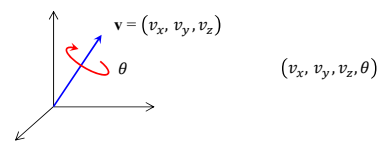
方位角の差が-180~180度になるように変換
前のスライドの式を記述
(単位はラジアン、M_PIはπ)

線形に補間

22

回転軸と回転角度による表現

• 回転軸と回転角度による向きの表現



- 任意の回転軸を用いることで、一つの回転のみで、どのような向きも表現できる
- 回転軸は長さ 1 の単位ベクトルとする
 $|v| = 1$

23

四元数による表現

- 単位四元数
 - 回転軸と回転角度による表現から変換

$$(v_x, v_y, v_z, \theta)$$

$$(x, y, z, w) = \left(v_x \sin \frac{\theta}{2}, v_y \sin \frac{\theta}{2}, v_z \sin \frac{\theta}{2}, \cos \frac{\theta}{2} \right)$$
- 単位四元数を使うメリット
 - 球面線形補間という、2つの向きの間を最短距離で補間する計算方法が使えるようになる

24

四元数

- 四元数 (Quaternion、クオータニオン)
 - 数学的には虚数を4次元に拡張したような概念

$$\mathbf{q} = (x, y, z, w) = xi + yj + zk + w = ((x, y, z), w)$$
 - 和、差、スカラー倍、共役 などの各種演算が定義できる
 - 向きを表す四元数は、単位四元数となる
 - 長さが1 $|\mathbf{q}| = \sqrt{x^2 + y^2 + z^2 + w^2} = 1$
 - 4次元空間での半径1の球面上の点として表せる → 球面上の最短経路上の点から向き補間を計算できる

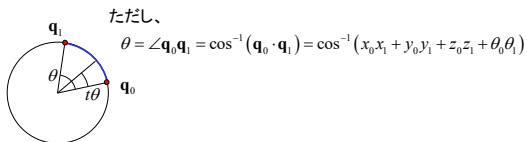


25

単位四元数の補間

- 球面線形補間 (SLERP: Spherical Linear Interpolation)
 - 四元数により表された2つの向きを補間

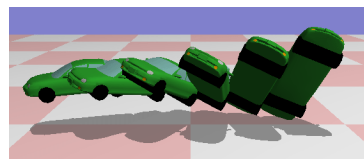
$$\mathbf{q} = \frac{\sin((1-t)\theta)}{\sin \theta} \mathbf{q}_0 + \frac{\sin t\theta}{\sin \theta} \mathbf{q}_1$$



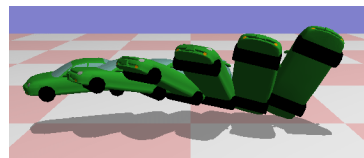
26

オイラー角による補間との比較

オイラー角



四元数



27

四元数と回転行列の変換(1)

- 単位四元数から回転行列への変換
 - 任意ベクトル周りの回転行列に相当

If the scalar part has value w , and the vector part values x , y , and z , the corresponding matrix can be worked out to be

$$M = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$$

when the magnitude $w^2+x^2+y^2+z^2$ equals 1. The

Ken Shoemake, "Animating Rotation with Quaternion Curves", Proc. of SIGGRAPH '85, pp. 245-254, 1985. より

28

四元数と回転行列の変換(2)

- 回転行列から単位四元数への変換
 - 回転行列の対角成分が回転角度を表す
 - ゼロ割を防ぐための特例を追加する必要がある

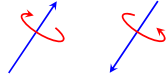
$$w^2 = 1/4 (1 + M_{11} + M_{22} + M_{33})$$

$w^2 > \epsilon ?$	
TRUE	FALSE
$w = \sqrt{w^2}$	$w = 0$
$x = (M_{23} - M_{32}) / 4w$	$x^2 = -1/2 (M_{22} + M_{33})$
$y = (M_{31} - M_{13}) / 4w$	$x^2 > \epsilon ?$
$z = (M_{12} - M_{21}) / 4w$	TRUE
	$x = \sqrt{-x^2}$
	$y = M_{12} / 2x$
	$z = M_{23} / 2x$
	FALSE
	$x = 0$
	$y^2 = 1/2 (1 - M_{33})$
	$y^2 > \epsilon ?$
	TRUE
	$y = \sqrt{y^2}$
	$z = M_{23} / 2y$
	FALSE
	$y = 0$
	$z = 1$

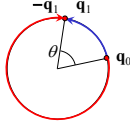
29

単位四元数の補間の注意

- 1つの向き表現方法は2通りある
 - (x, y, z, w) と $(-x, -y, -z, -w)$ は共役解



- 向きの間を補間する際は、通常、角度が小さくなる方の共役解を使用する



30

プログラム例

- 四元数による向きの補間の処理の流れ

```
// 区間の両端点の向きを取得
const Matrix3f & o0 = keyframes[ seg_no ].ori;
const Matrix3f & o1 = keyframes[ seg_no + 1 ].ori;

// 行列による向きの表現を四元数による表現に変換
Quat4f q, q0, q1;
q0.set( o0 ); q1.set( o1 );

// 2つの四元数の間の角度が90度以上あれば、共役の四元数を使用
if ( q0.x * q1.x + q0.y * q1.y + q0.z * q1.z + q0.w * q1.w < 0 )
    q1.negate( q1 );

// 球面線形補間を計算
...
// 計算後の四元数を行列表現に変換
```

球面線形補間(前のスライドの式)を自分で計算 or vecmath の Quat4 クラスのメソッド (interpolate()) を使って計算

31

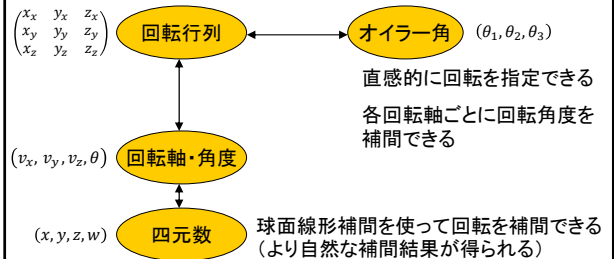
向きの表現方法のまとめ

- 向きの表現方法
 - 回転行列による表現 (3x3行列) $\begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$
 - 基本的な表現方法
 - 余計なデータが多い、補間は難しい
 - オイラー角による表現 $(\theta_1, \theta_2, \theta_3)$
 - 人間にとって記述がしやすい
 - 各回転軸ごとに回転角度を補間できる
 - 回転軸と回転角度による表現 (v_x, v_y, v_z, θ)
 - 四元数による表現 (x, y, z, w)
 - 球面線形補間を使って向きを全体的に補間できる

32

向きの表現方法と相互変換

- 回転行列による表現方法が基本



33

向きの表現方法の決定

- 自分のプログラムでどのような表現方法を用いるか?
 - どちらにしても、描画のため、最後は回転行列の形にする必要がある
 - 方法1: オイラー角または四元数として扱い、最後だけ回転行列に変換
 - 方法2: 回転行列として扱い、必要に応じて四元数やオイラー角に変換
 - 視点操作の回、変換行列を使う方法とパラメータ表現(オイラー角)を使う方法の使い分けと同じ

34

キャラクタ・アニメーション

- 人体を多関節体として扱い、各関節の回転によって姿勢を表現する
 - 関節の回転の表現方法
 - 昔はオイラー角が一般的に使われていた
 - キーフレームアニメーションを行ったときに関節の回転が不自然になる
 - 最近は回転行列・四元数による表現が一般的に使われている
 - 基準部位(腰)の位置も必要
 - 詳細は、後日の講義で説明



35

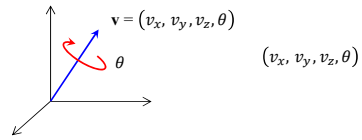
補足: 複数の回転の補間

- 用途によっては、3つ以上の向き・回転を補間する必要がある
 - 動作補間により、複数の動作データを混合して新しい動作を生成する場合など
- 四元数を使った球面線形補間では、2つの向き・回転の補間しかできない
- 対数ベクトル表現を使うことで複数の向き・回転の補間が可能

36

対数ベクトル表現

- 回転軸と回転角度による向きの表現

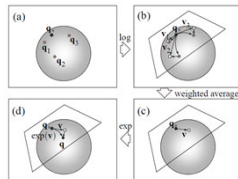


- 単位四元数 $(v_x \sin \frac{\theta}{2}, v_y \sin \frac{\theta}{2}, v_z \sin \frac{\theta}{2}, \cos \frac{\theta}{2})$
- 対数ベクトル $(v_x \frac{\theta}{2}, v_y \frac{\theta}{2}, v_z \frac{\theta}{2})$

37

対数ベクトル表現による補間

- 四元数を対数ベクトル表現に変換
- 対数ベクトルの線形補間により、複数の回転を補間できる
 - 単純に補間すると誤差が大きくなる
 - 平均回転 q^* を求めて、それと各回転の差分を表すベクトルを補間



Sang Il Park, Hyun Joon Shin, Sung Yong Shin, "On-line locomotion generation based on motion blending", ACM SIGGRAPH Symposium on Computer Animation 2002, pp. 105-111, 2002.

38

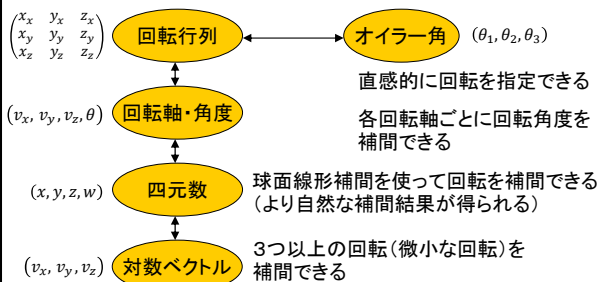
向きの表現方法のまとめ

- 向きの表現方法
 - 回転行列による表現 (3x3行列) $\begin{pmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{pmatrix}$
 - 基本的な表現方法、補間は難しい
 - オイラー角による表現 $(\theta_1, \theta_2, \theta_3)$
 - 人間にとって記述がしやすい
 - 各回転軸ごとに回転角度を補間できる
 - 回転軸と回転角度による表現 (v_x, v_y, v_z, θ)
 - 四元数による表現 (x, y, z, w)
 - 球面線形補間を使って向きを全体的に補間できる
 - 対数ベクトルによる表現 (x, y, z)
 - 3つ以上の回転の補間(微小な回転の補間)

39

向きの表現方法と相互変換

- 回転行列による表現方法が基本



40

今日の内容

- 向きの補間
 - オイラー角
 - 四元数と球面線形補間
 - 相互変換

- アニメーションプログラミング
- レポート課題

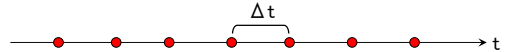
41

アニメーションプログラミング

42

アニメーションプログラミング

- アニメーション速度を一定に保つための工夫
 - アニメーション処理(アイドル処理)が一定周期で実行される保証はない
 - アイドル処理が呼ばれる毎に一定時間アニメーションを進めるような単純なプログラムでは、コンピュータの性能や画面サイズなどにより、アニメーションの速度が大きく変わってしまう
 - 描画速度に合わせて、アニメーションの速度を自動的に調節するような工夫が必要

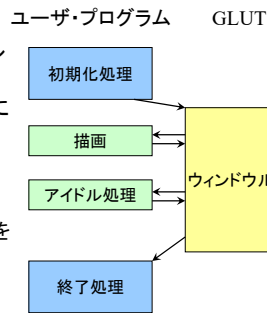


43

GLUTのイベントモデル(復習)

• イベントドリブン

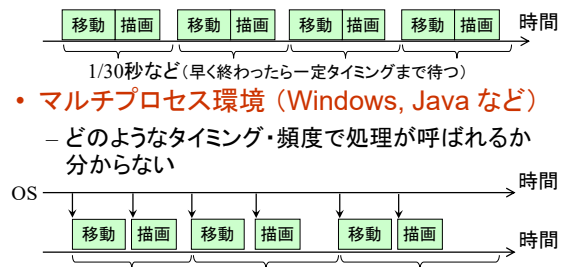
- 描画処理やアイドル処理を設定しておくことで、必要なときにそれらが呼ばれる
- アイドル処理は、定期的に呼ばれる
 - アニメーション処理をここに記述
 - どれくらいの頻度で呼ばれるかは不明



44

アニメーションの処理

- 非マルチプロセス環境(ゲーム専用機など)
 - 常に一定のタイミングで処理ができる
- マルチプロセス環境 (Windows, Java など)
 - どのようなタイミング・頻度で処理が呼ばれるかわからない

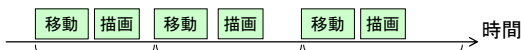


45

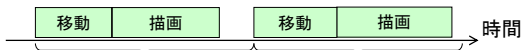
再生速度の問題

- 1度のアニメーション処理の度に一定量移動を行う、というプログラムになっていると・・・
- アニメーション処理が呼び出される頻度によって、移動速度が異なってしまう

実行回数が多いので、結果的に沢山移動



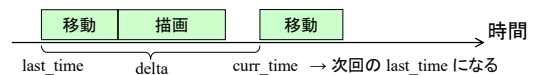
実行回数が少ないので、結果的に少しか移動



46

再生速度を一定にする工夫

- アイドル関数(移動処理)での移動量を調整
 - 現在の時刻を取得 → `curr_time`
 - 前回呼ばれたときの時間の差を計算
`delta = curr_time - last_time;`
 - `delta` の大きさに合わせて、物体を動かす
 - 今回の時刻を記録
`last_time = curr_time;`
 - `last_time` は、静的である必要がある



47

時刻の取得

- C標準関数
 - time() 秒単位の精度でしか取得できない
 - clock() CPU時間を取得、CLOCKS_PER_SEC で割る
- Windows API 関数
 - timeGetTime() OS起動時からの経過時間を取得
 - C標準関数よりも高い精度
 - 古い Windows (95など) では、精度が悪い(10ミリ秒程度)ので、timeBeginPeriod() で調整
 - QueryPerformanceCounter() CPUのクロックカウンタにもとづいた高精度な経過時間を取得可能

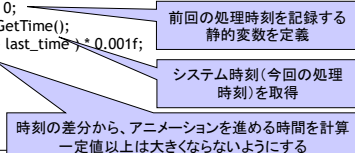
48

プログラム例

- サンプルプログラムでのアニメーション処理

```
// アニメーションの再生時間
float animation_time = 0.0f;

// アイドル時に呼ばれるコールバック関数
void IdleCallback( void )
{
    // システム時間を取得し、前回からの経過時間に応じてΔtを決定
    static DWORD last_time = 0;
    DWORD curr_time = timeGetTime();
    float delta = ( curr_time - last_time ) * 0.001f;
    if ( delta > 0.1f )
        delta = 0.1f;
    last_time = curr_time;
    animation_time += delta;
}
```



49

まとめ

- キーフレームアニメーションの基礎
- サンプルプログラム
- 行列・ベクトルを扱うプログラミング
- 位置補間
 - 線形補間、Hermite曲線、Bézier曲線、B-Spline曲線
- 向きの補間
 - オイラー角、四元数と球面線形補間、相互変換
- アニメーションプログラミング
- レポート課題

50

レポート課題

- 位置・向き補間を実現するプログラムを作成
 1. Hermite曲線による位置補間
 2. Bézier曲線による位置補間
 3. B-Spline曲線による位置補間
 4. 四元数と球面線形補間による向き補間
 - サンプルプログラム (keyframe_sample.cpp) をもとに作成したプログラムを提出
 - 他の変更なしのソースファイルやデータは、提出する必要はない
 - Moodleの本講義のコースから提出
 - 締切: Moodleの提出ページを参照

51

レポート課題 提出方法

- Moodleから、以下の2つのファイルを提出
- 作成したプログラム(テキスト形式)
 - keyframe_sample.cpp
 - 変更箇所のみを抜き出したレポート(PDF)
 - Moodle に公開している LaTeX のテンプレートを もとに、作成する
 - これまでのレポートと同様

52

レポート課題 演習問題

- レポート課題の提出に加えて、レポート課題の理解度を確認するための Moodle 演習問題にも解答する
 - 解答締切は、レポート提出と同じ
 - 締切までは解答を変更可、締切後に正解が表示
 - レポート課題のヒントにもなっているので、レポート課題で分からない箇所があれば、演習問題の説明・選択肢を参考にして考えても良い
 - 本演習問題の点数は、演習課題の成績の一部として考慮する

53

次回予告

- **物理シミュレーション**
 - 物理シミュレーションの種類
 - 剛体の物理シミュレーション
 - 運動方程式
 - 回転運動と慣性モーメント
 - シミュレーションの手順
 - 衝突と接触の扱い
 - 多関節体・変形する物体のシミュレーション

54

演習問題

- **Moodle の演習問題を受験する**
 - 指定された期限までに解答する
- **今回は、Moodle のプログラミング演習問題も受験する**
 - VPL (Virtual Programming Lab) の機能を利用
 - 指定された処理を実現するように、与えられたプログラムの空欄を記述する
 - VPL 上で、コンパイル・実行、評価を行う
 - 評価まで行い、正しく実行されることを確認

55