KAIST

CS 380

Introduction to Computer Graphics

LAB (6)

2018.04.30

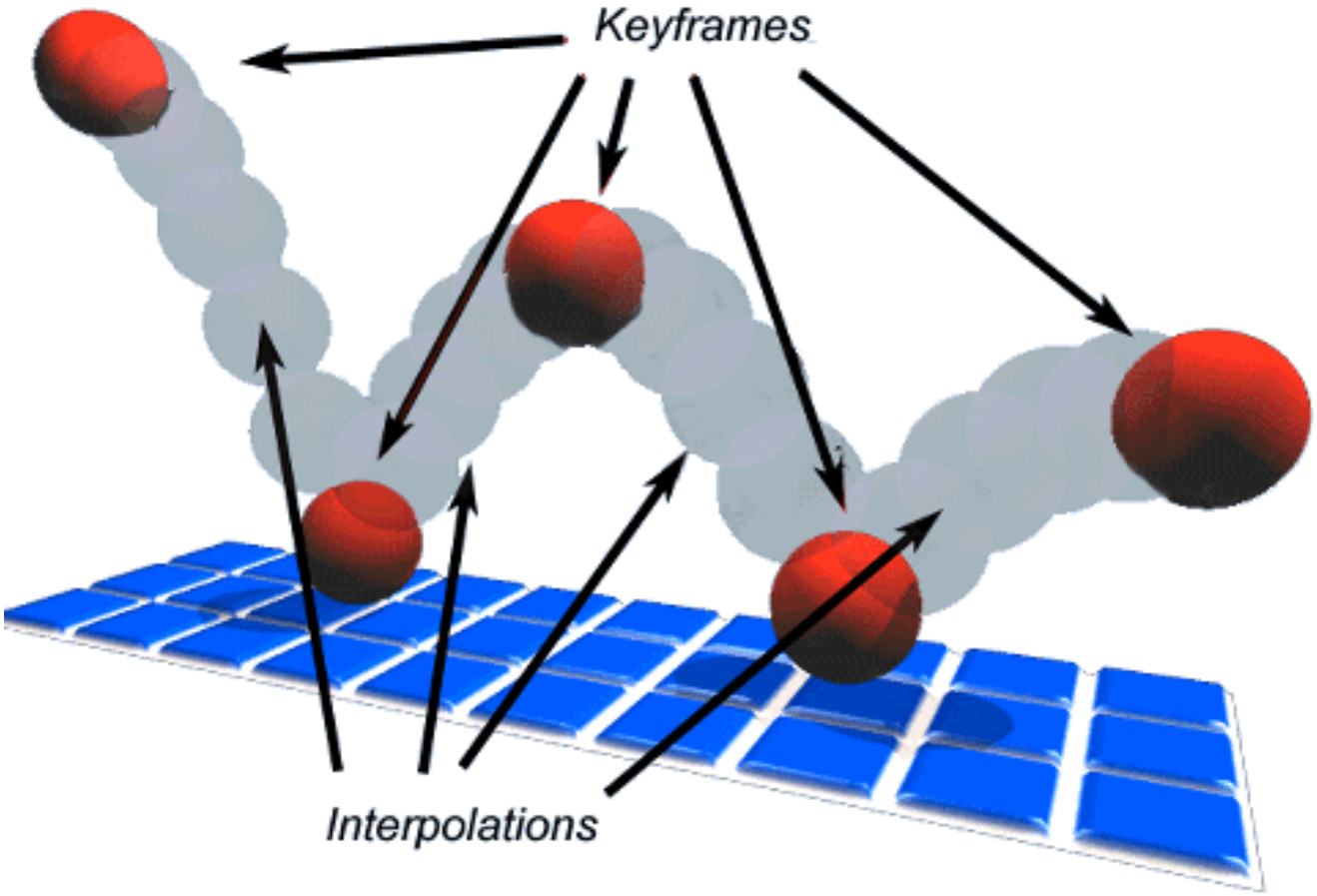
VISUAL COMPUTING Lab

Tasks

KAIST

- Keyframes
- Linear Interpolation
- Playing the Animation

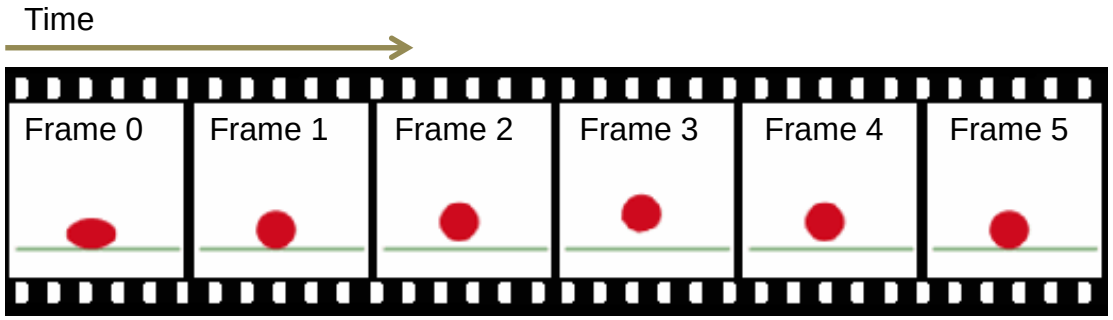
Keyframe Animation

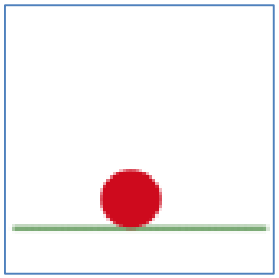


Animation

KAIST

- A sequence of frames
  - To depict motion and shape change



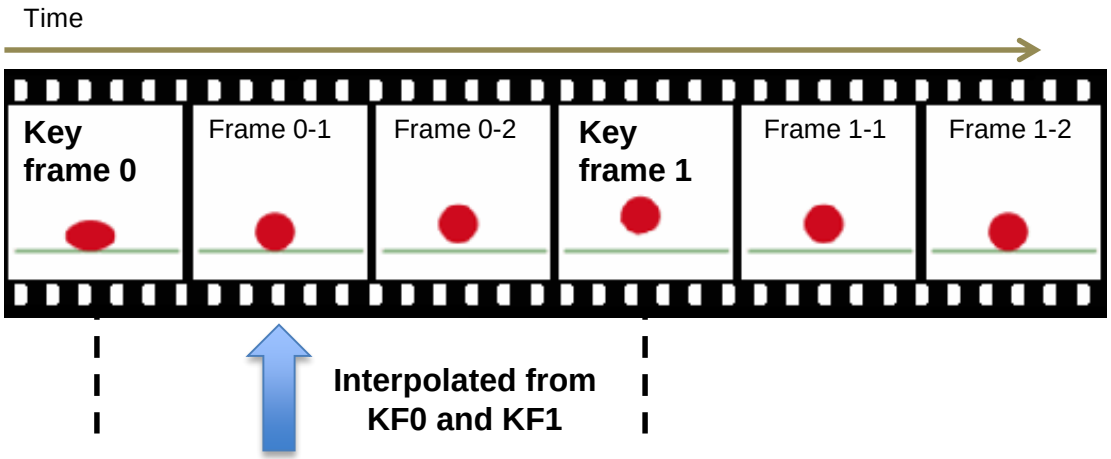


- Frame (in animation)
  - Like a video frame which is a image at one instant.
  - Here, the state of scene (geometric relation represented by a scene graph) at one instant.
  - Do not confuse with the definition of frame in graphics

Key frame sequence

KAIST

- Key frame
  - Storing all frames is memory-consuming.
  - Method: Just store **a few important frames** and interpolate intermediate frames from key frames.

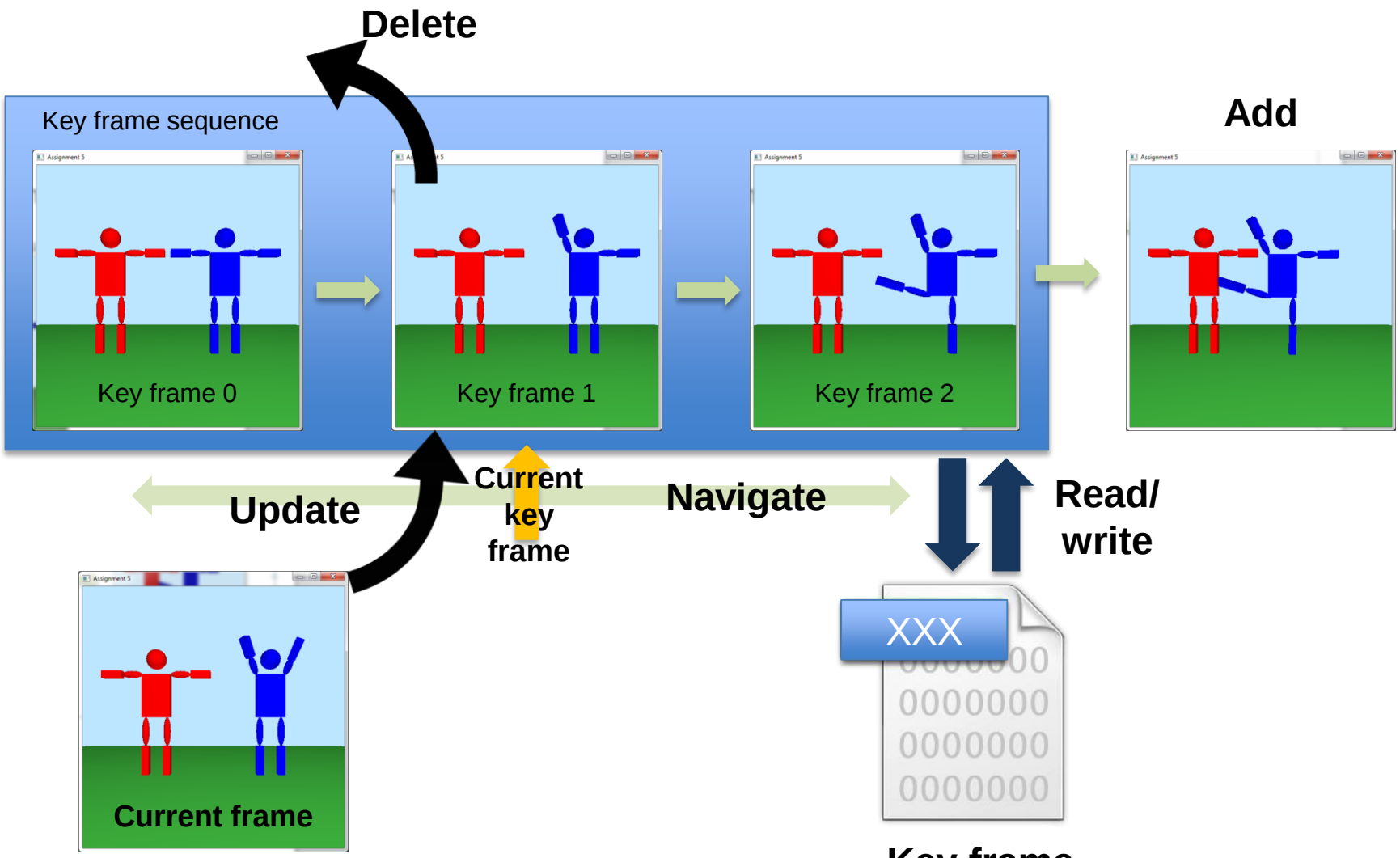


- Key frame sequence
  - A sequential list of key frames

Task1

- You should implement keyframe
- Store keyframes as a list
- Each keyframe contains **RigTForm**s of all transformation nodes in a scenegraph.

Overview for task 1



Hotkeys Implementation

KAIST

- Space: Show current keyframe
- u: Update current keyframe
- >: Move to next keyframe
- <: Move to previous keyframe
- d: Delete current keyframe
- n: Create a new keyframe
- i: read keyframes file
- w: write keyframes file

Vector vs. List in C++ STL

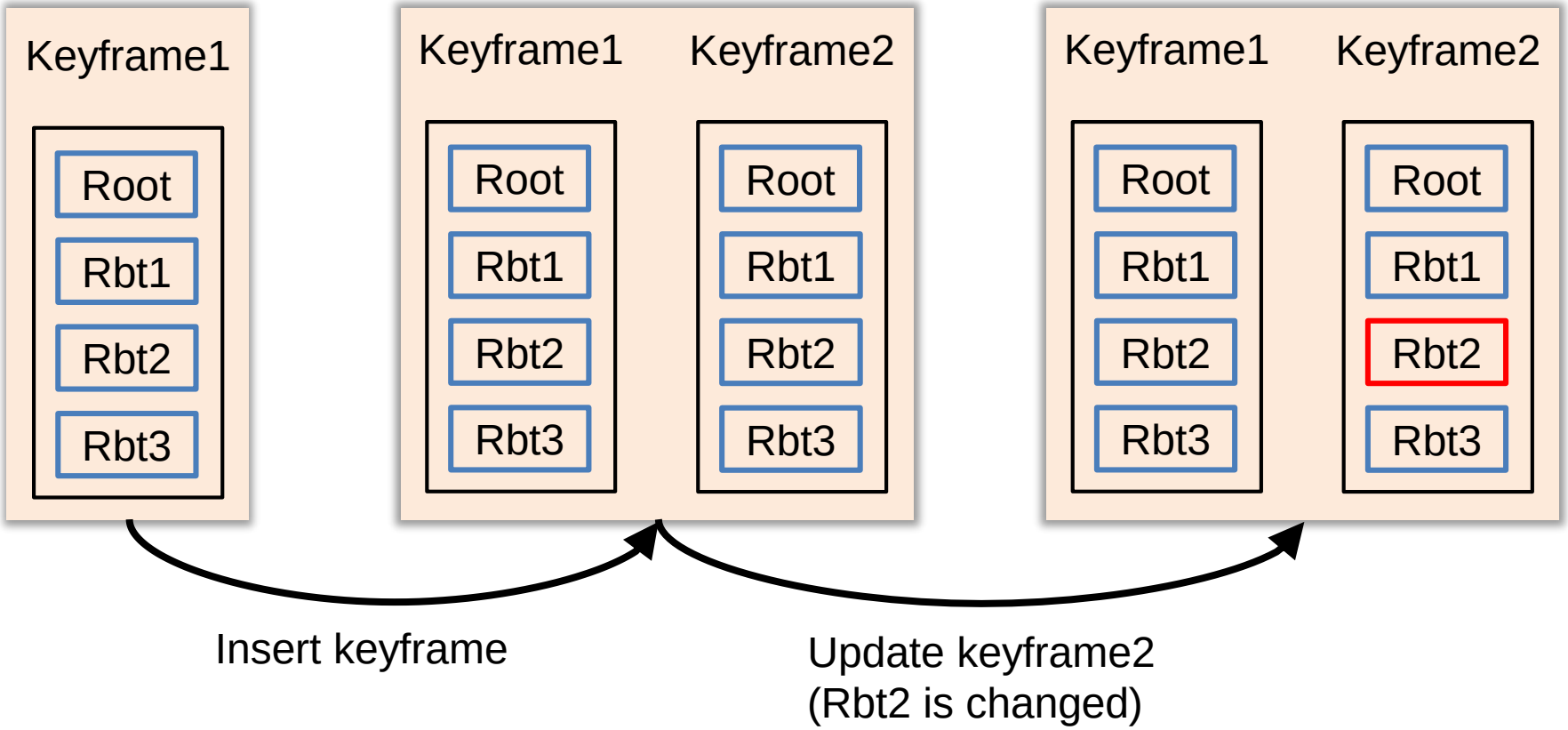
KAIST

- Same functionality, but different implementation.
- Not a big issue whether use vector or list
- List is faster when add/delete elements. (read appendix in the spec document)
- We recommend you to use list for storing keyframes

Adding Keyframe

KAIST

- An example of keyframe implementation
- The structure is a list of vectors



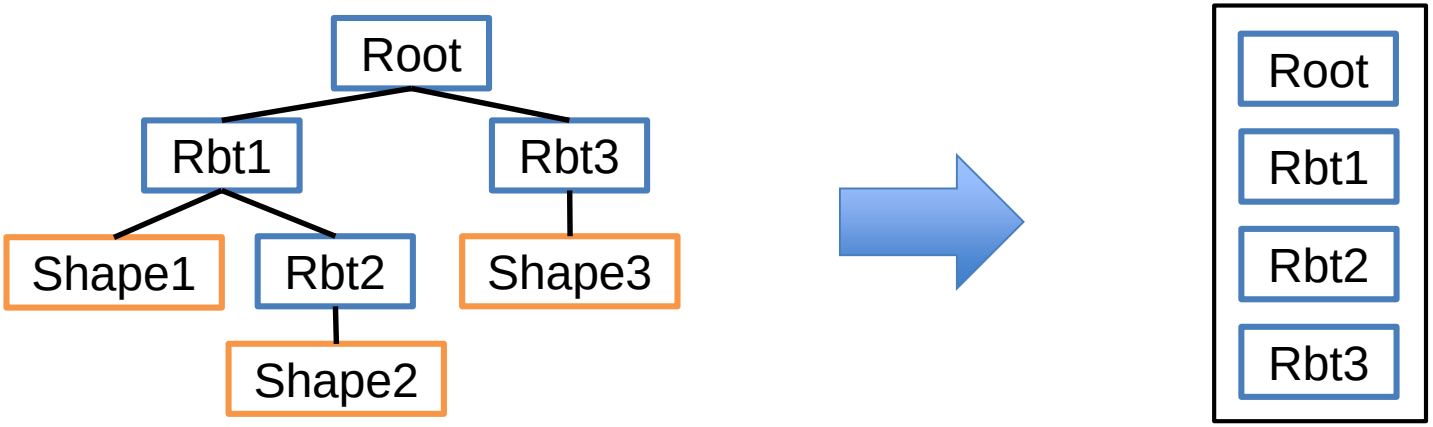
VISUAL COMPUTING Lab

dumpSgRbtNodes

KAIST

- Two paramters (*SgNode, vector<*SgRbtNode>)
  - First parameter (*SgNode): Root node of the scenegraph
  - Second parameter (vector<*SgRbtNode>): Empty vector for storing rbt nodes

- After you call this function, all RbtNodes in the scenegraph are stored in a vector (second parameter)



Keyframes File

KAIST

- When you press ‘w’, a keyframes file should be created
- The keyframes file should include all rbt information (translation, rotation) of each keyframe.
- When you read the keyframes file by pressing ‘i’, the saved keyframes should be loaded.

File Write/Read

KAIST

Using file stream is just one example. You can use any read/write method.

- Write file using stream

```
ofstream f(filename, ios::binary);
f << numKeyframes << ‘ ‘ << numRbtNodes << ‘\n’;
… # write each Rbt to a file
```

- Read file using stream

```
const char *filename;
int numFrames, numRbtsPerFrame;
ifstream f(filename, ios::binary);
f >> numKeyframes >> numRbtNodes;
… # read each Rbt from a file
```

Task2

- Vector interpolation

$$c = (1-\alpha)c_0 + \alpha c_1$$

- Quaternion interpolation
  - cn indicates **conditional negate** operation

$$q = \left(\mathrm{cn}\left(q_1 q_0^{-1}\right)\right)^a q_0$$

Quaternion Power Function

KAIST

- Quaternion power function needs to be implemented for spherical interpolation
- Firstly, rotation angle should be calculated
- Becareful that arccos and arcsin cannot determine the correct ϕ in $0 \sim 2\pi$

$$p = \cos(\phi) = \cos(-\phi) \qquad \arccos(p) = \phi \text{ or } 2\pi - \phi$$

$$q = \sin(\phi) = \sin(\pi - \phi) \qquad \arcsin(q) = \phi \text{ or } \pi - \phi$$

- You may use “atan2” function.

$$\text{atan2}(\sin(\phi), \cos(\phi)) = \phi$$

RigTForm Interpolation

KAIST

- RigTform consists of a translation (3d vector and) rotation (quaternion)
- Interpolate each values

```
rbt.t = lerp(rbt1.t, rbt2.t, alpha);  # implement vector interpolation
rbt.r = slerp(rbt1.r, rbt2.r, alpha);  # implement quaternion interpolation
```

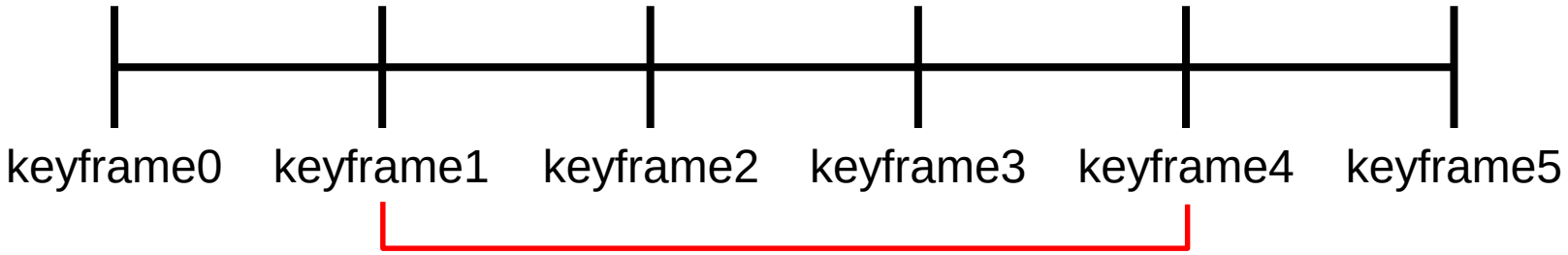
Task3

KAIST

- You needs to implement animation
- The animation interpolates each pair of keyframes

- Hotkeys to be implemented
  - “y”: play/stop the animation
  - “+”: make animation faster
  - “-” : make animation slower

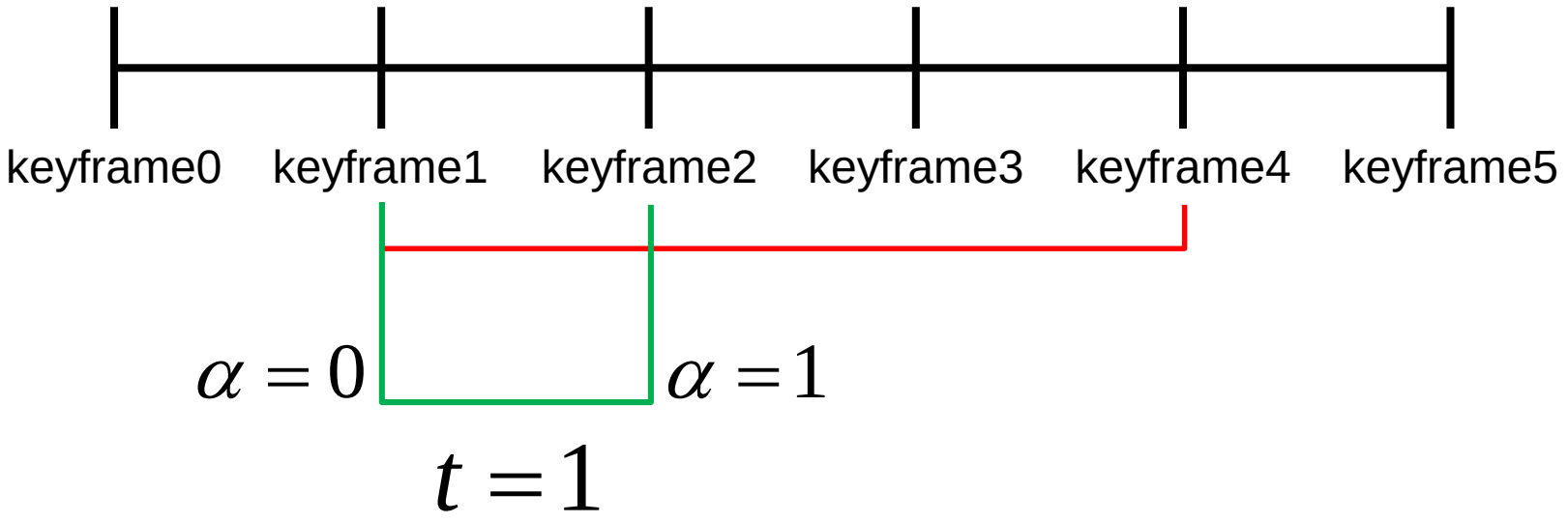
Animation Implementation



Play animation from the **second keyframe** to **second to last keyframe**.

Exclude the first and the last keyframe for next assignment
(Catmull-Rom interpolation)

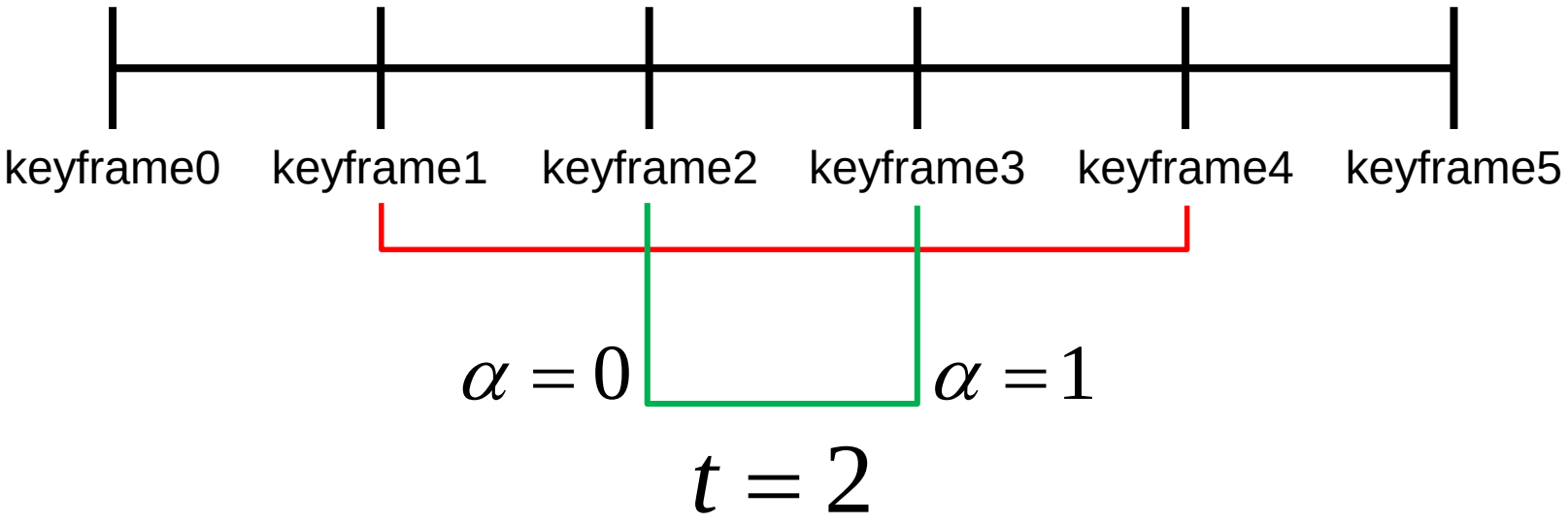
Animation Implementation



- Animate using two keyframes: **keyframe1** and **keyframe2**
- Continuously increase interpolation factor α.

t : current keyframe index

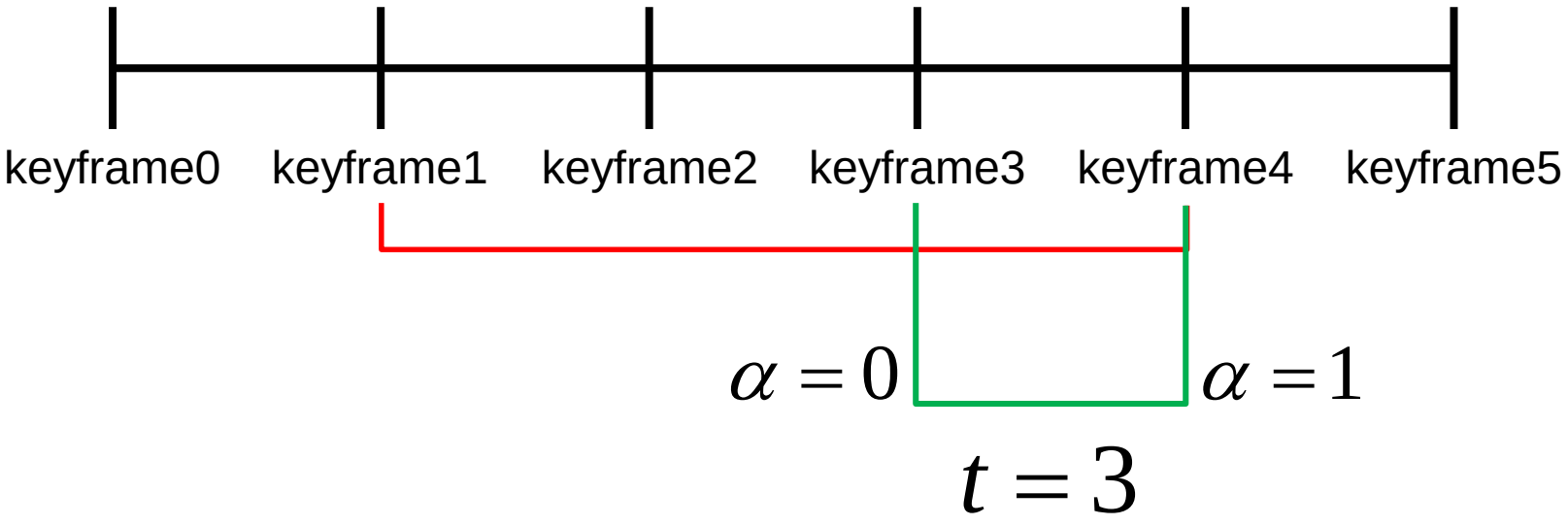
Animation Implementation



- When the interpolation between two keyframes is finished, reset α to zero and increase the current keyframe index t.

t : current keyframe index

Animation Implementation



- Keep doing same thing until the whole animation is finished

t : current keyframe index

glutTimerFunc

KAIST

- Three paramters
  - First: Time to call function
  - Second: Function to be called
  - Third: parameter for the function

- Example

```
void timerCallback(int next_ms) { … }
…
glutTimerFunc(time_ms, timerCallback, next_ms);
```

  - The timerCallback function will be executed in time_ms later with parameter next_ms

glutTimerFunc

KAIST

- Why we use glutTimerFunc?
  - Controller the animation timing
  - You can call a function periodically with specific timer interval
- Don’t forget to call “glutPostRedisplay()” when something needs to be redrawn.

Submission

KAIST

- Homework due
  - 5/9 (Wed) 23:59
- Submission
  - Zip file name: hw1_20161234_Name.zip
- Your zip file should include at least one animation file
  - A file format does not matter. However, it should be playable in your program.