CS30800
Introduction to Computer Graphics
Lab 6 – Animation

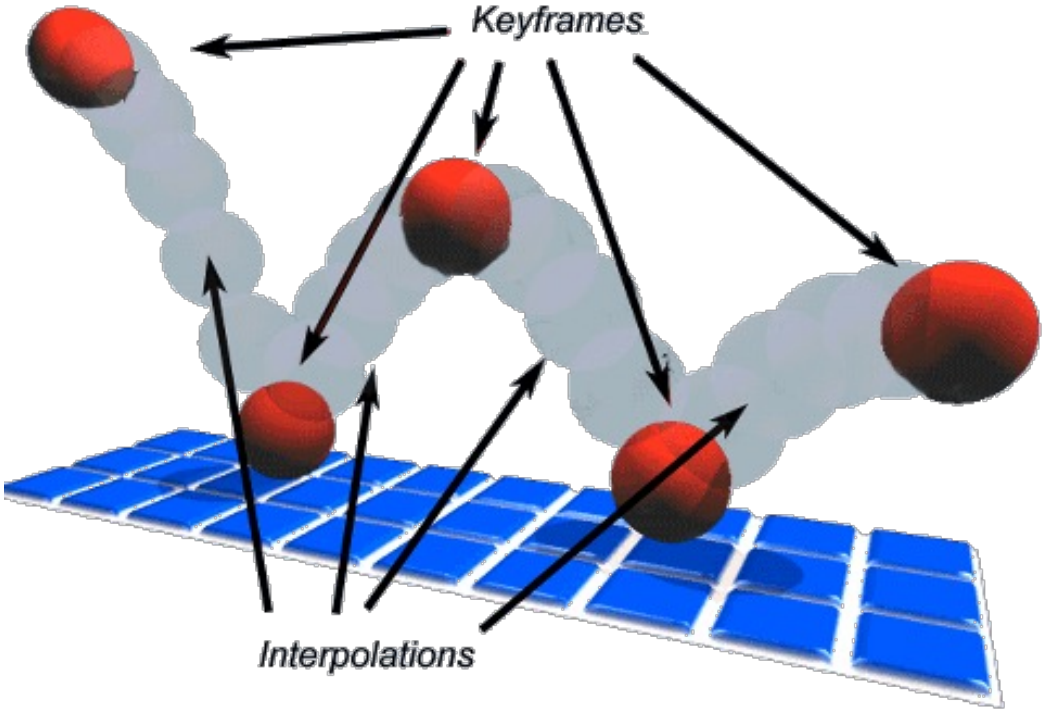
2025. 04. 22 / 2025. 04. 24

Tasks

- Add key frame feature
- Linear interpolation
- Playing animation

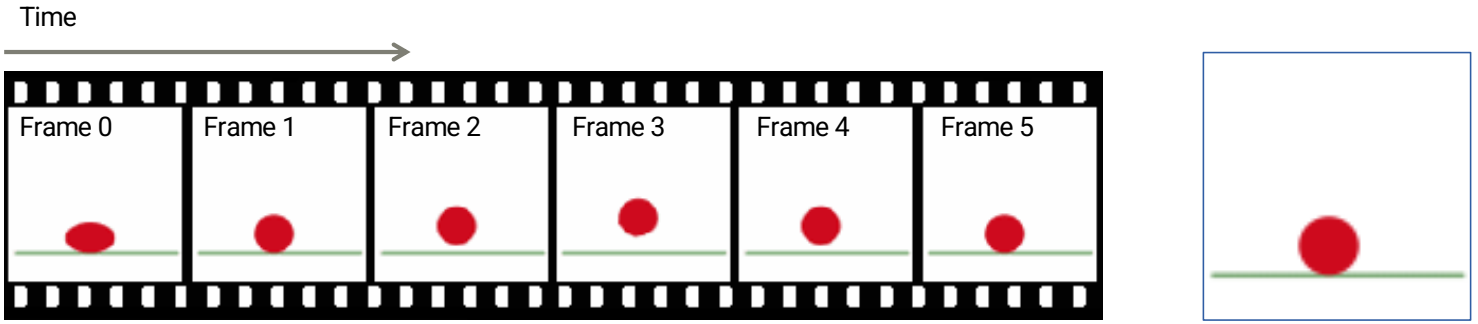
Introduction

Key Frame in Animation



Animation

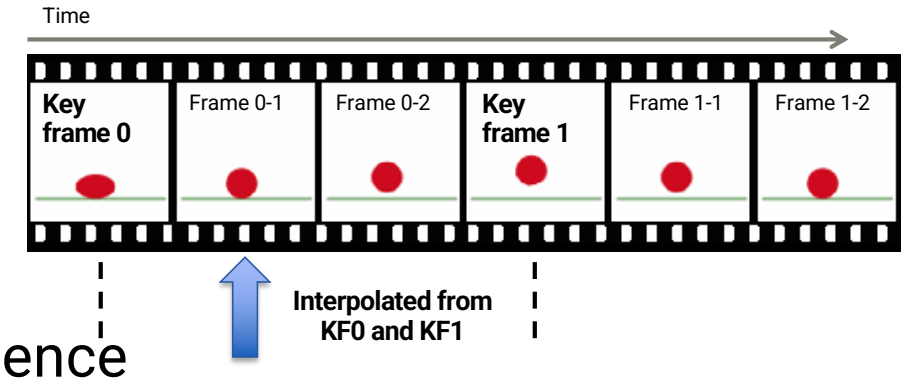
- A sequence of frames
  - Describe the motion and shape changes of the object



- Frame (in animation)
  - Represents the appearance of the object at specific time
  - Do not confuse with the frame $\vec{f}^t$ in graphics

Key frame sequence

- Key frame
  - Storing all frames is memory-consuming.
  - Storing only important frames: key frame
  - Interpolate intermediate frames using key frames.



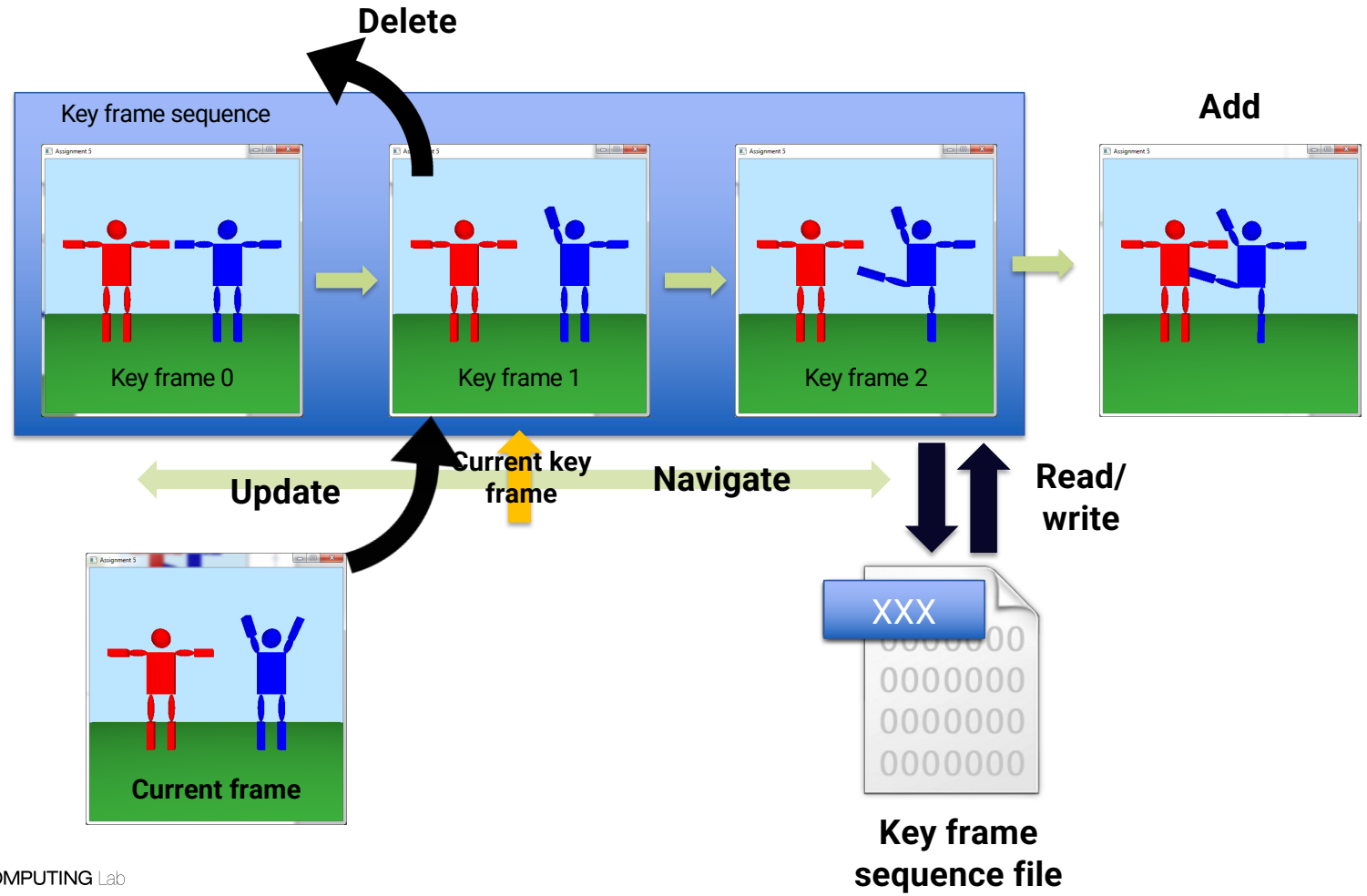
- Key frame sequence
  - A sequence of key frames which describes motion and deformations of the s cene.

HW5 Task 1 – Key Frame I/O

Task 1

- You should implement keyframe feature.
- Each key frame can be described by using RigTForms.
- Store key frame using `std::list`
- Each keyframe contains RigTForms of all nodes in a scenegraph.

Task 1


Delete
Key frame sequence
Add
Key frame 0
Key frame 1
Key frame 2
Update
Current key frame
Navigate
Read/ write
Current frame
XXX
Key frame sequence file

Hotkeys Implementation

- ‘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 from file
- ‘w’: write keyframes file

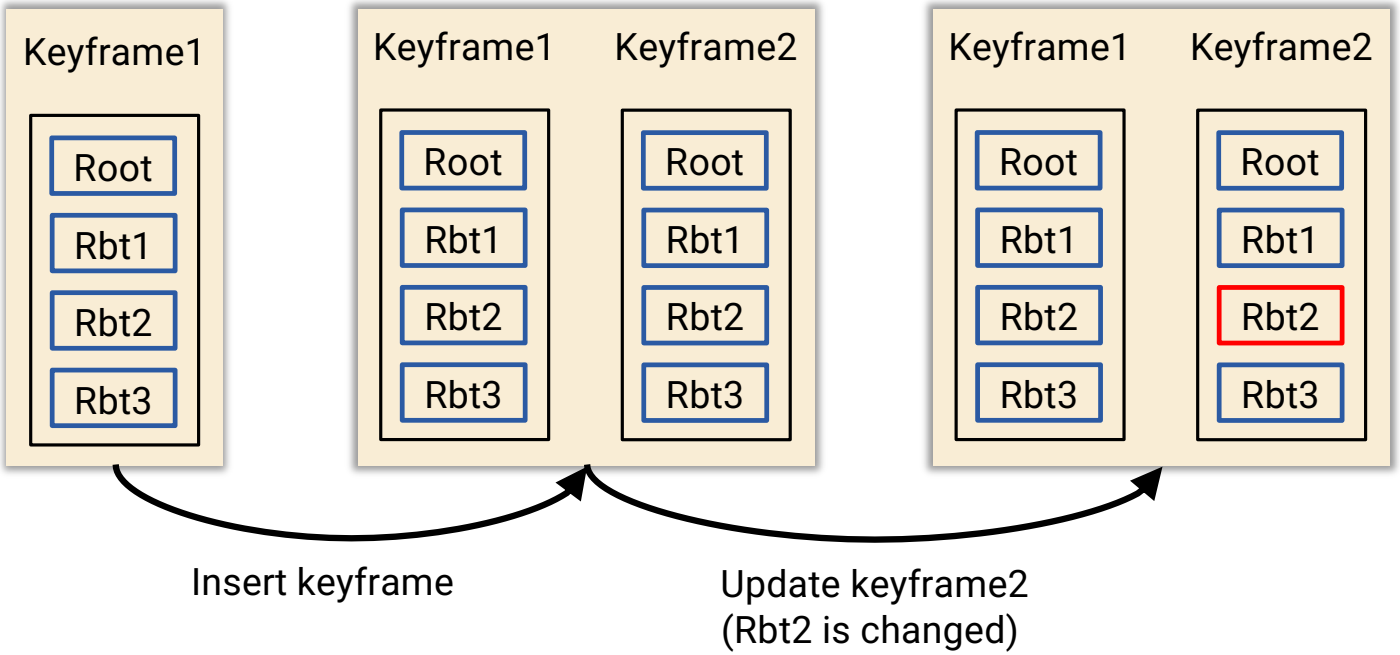
C++ STL (Standard Template Library) – Vector vs. List

- std::vector
  - Dynamic Array
- std::list
  - Double Linked List

- Pros and cons
  - Vector: Insert/delete – O(n), random access O(1)
  - List: Insert/delete – O(1), random access O(n)
  - Frequent add/delete → List, Fixed number of elements → vector

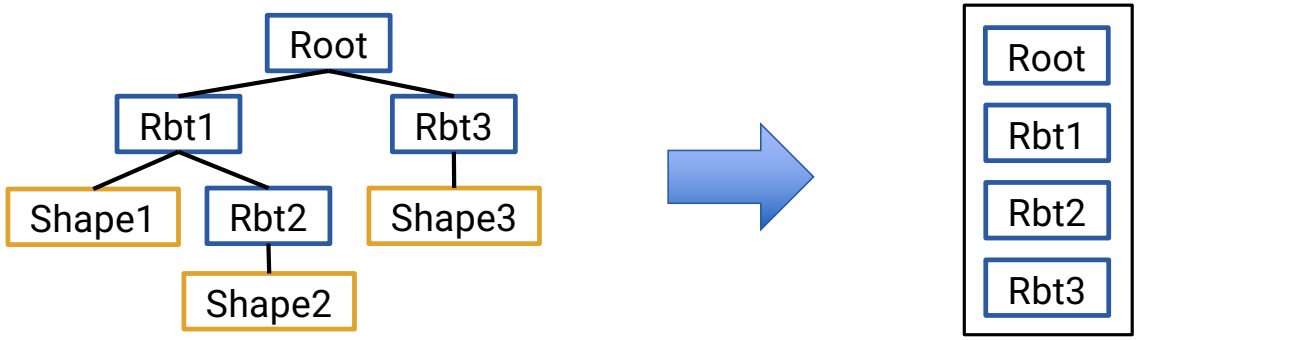
Add Key Frame

- Manage Rbts in each keyframe using std::vector
- Manage Keyframes using std:::List



dumpSgRbtNodes

- Two parameters (*SgNode, vector<*SgRbtNode>)
  - First parameter (*SgNode): Root node of the scenegraph
  - Second parameter (vector<*SgRbtNode>): Vector for storing RbtNodes
- After the function call, all RbtNodes in the scenegraph are stored in a vector



Read / Write Key Frames

- Press ‘w’ to export key frames as a file.
- The key frames file should contain all Rbt information (translation, rot ation) for each key frame.
- Press ‘i’ to import key frames from a file.
- The key frames should be restored properly.

File Read / Write

- You can use any kind of file I/O functions to load / store Rbts.
- Below are one of the example.

- Write file using ofstream

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

- Write file using ifstream

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

HW5 Task 2 – Key Frame Interpolation

Task 2 - Interpolation

- Vector interpolation

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

- Quaternion interpolation
  - cn denotes conditional negation to select short interpolation

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

RigTForm Interpolation

- RigTForm
  - Translation (3D vector)
  - Rotation (Quaternion)

- Interpolate translations and rotations

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

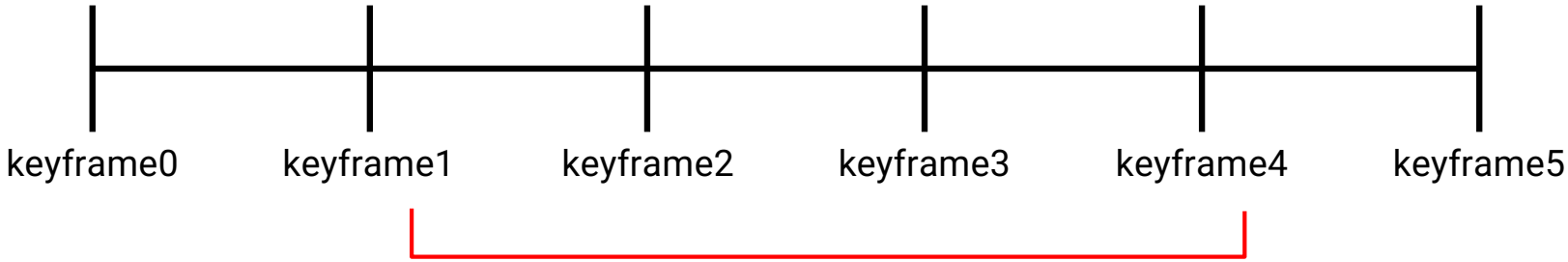
HW5 Task 3 – Animation

Task 3 – Animation

- You need to implement the animation playback feature.
- Animation visualizes interpolated frames between two keyframes.

- Hot keys for animation playback
  - ‘y’: play/stop the animation
  - ‘+’: make play speed faster
  - ‘-’: make play speed slower

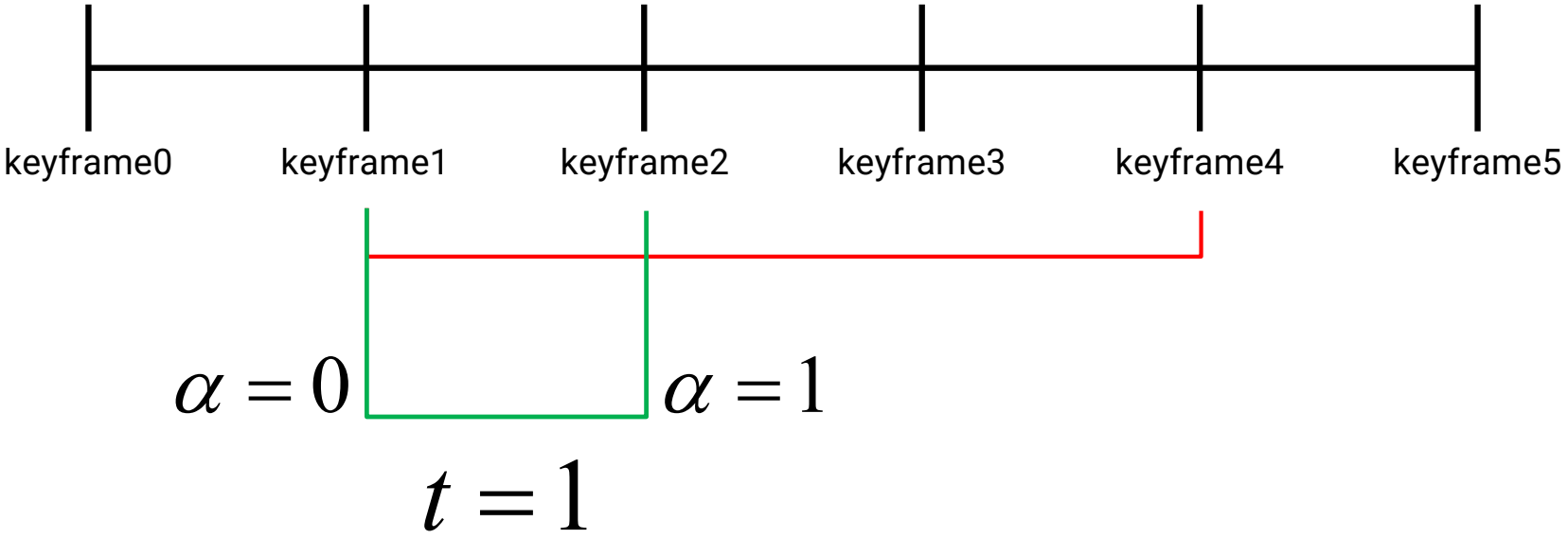
Animation Implementation



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

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

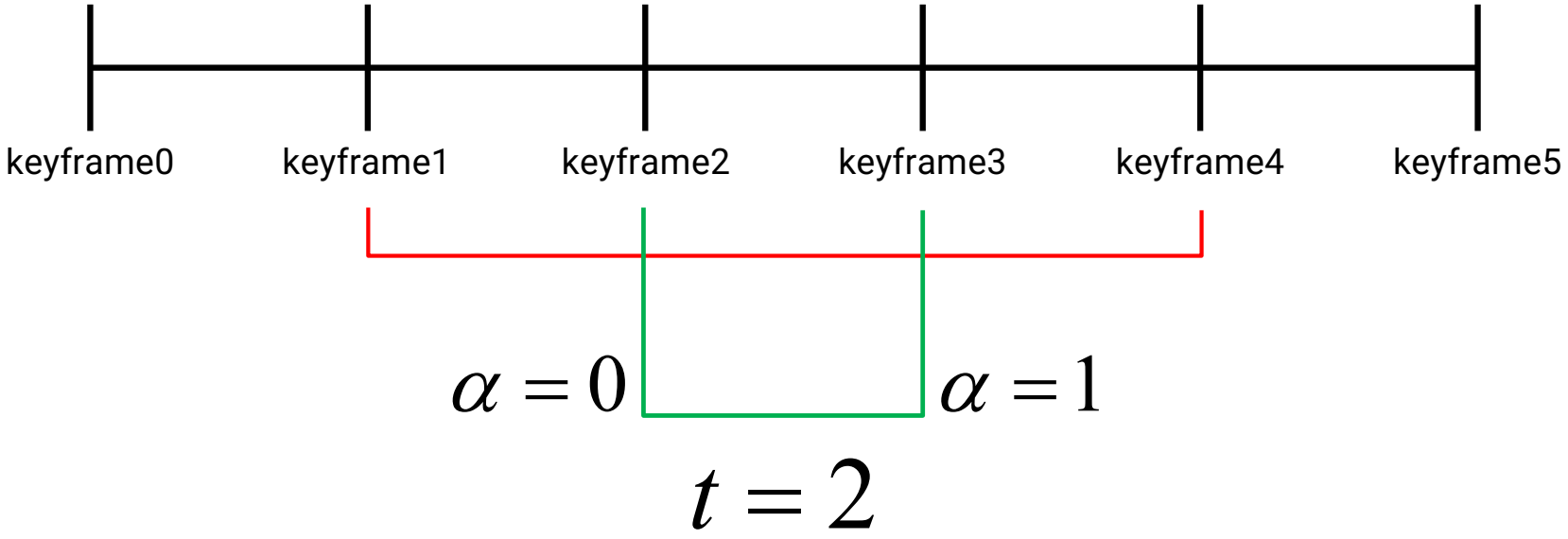
Animation Implementation



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

t : current keyframe index

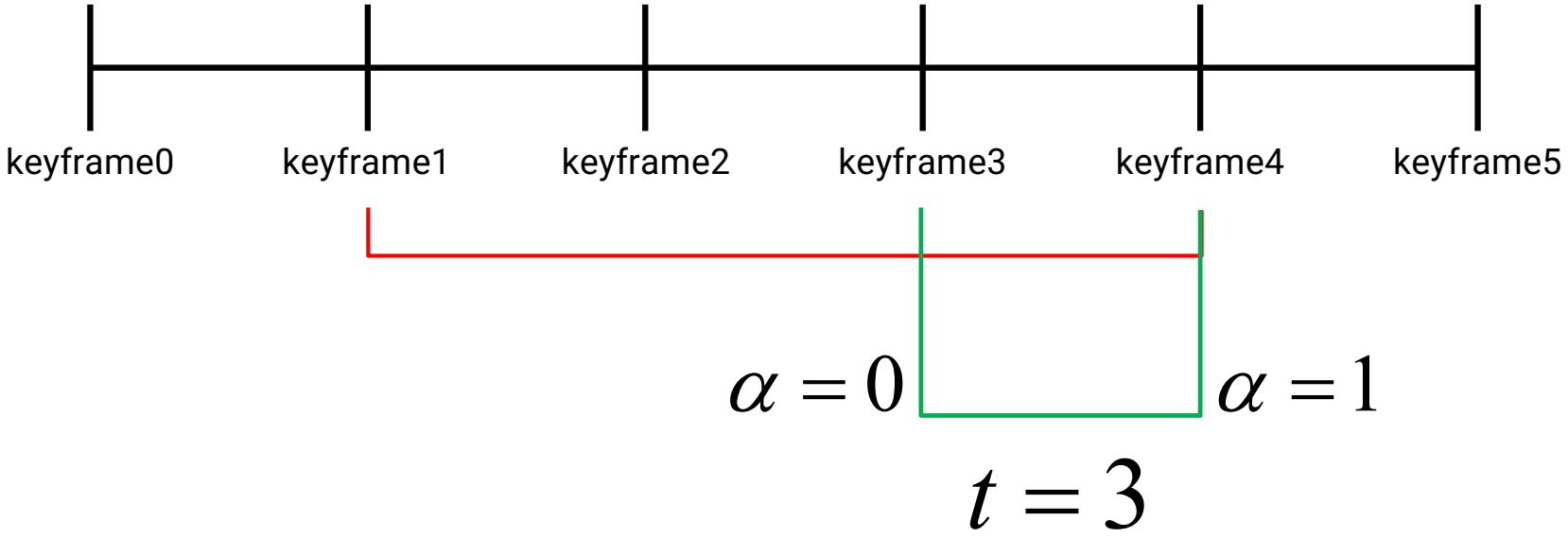
Animation Implementation



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

t : current keyframe index

Animation Implementation



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

t : current keyframe index

glutTimerFunc

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

- Example
  - The timerCallback function will be executed after time_ms with parameter next_ms

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

glutTimerFunc

- Why we need to use this function?
  - To control the animation playback speed
  - You can call the function periodically with specific time interval

- Don’t forget to call “glutPostRedisplay()” after update Rbts.

Submission

- Homework deadline:
  - 5/4 (SUN) 23:55

- Submission
  - Zip file name: hw5_2025xxxx_{name}.zip

- **Your zip file must include at least one animation file!!**
  - Since the scenegraph implementation varies each other.