CASE HW2

309553008

1. Introduction
   - 了解 asf 及 amc 檔的資料內容與結構
   - 把 local space 的座標轉到 world space
   - 將想要呈現的 key frame 放到想要出現的位置(秒數) ，然後利用內插產生需要的 frame，以調整整個動畫
2. Fundamentals
   - global coordinate：

     整個場景的座標，也就是 world space，所有物體最終的位置座標，是絕對座標而不是相對座標，所以是不會動的
   - local coordinate：

     每個物件有自己的 local coordinate，當 translate、rotate、scale 物體時，local coordinate 會改變。存在的目的是讓使用者更容易的操作不同的物件，不用去考慮其他物體對現在操作的物體有什麼影響。
3. Implementation
   - forwardSolver:

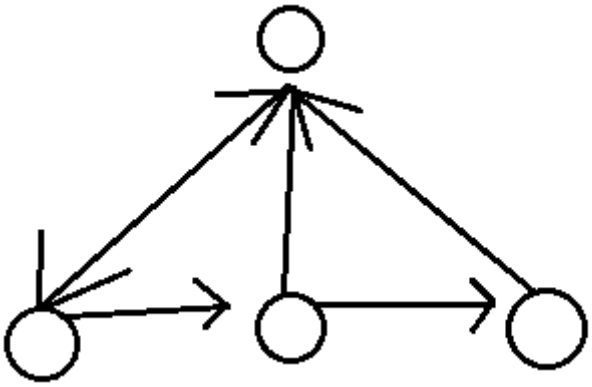
     先把 root 定好，因為是採用 BFS，所以設定好 root 後，在 BFS 的 mask 也做上標記。而 root 沒有 parent，所以不需要乘上 rot_parent_current，也沒有長度，所以也不需要算 rotation*length

```
bool visited[30] = {};
// root
bone->start_position = posture.bone_translations[0];

bone->rotation = bone->rot_parent_current;
bone->rotation = util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);

bone->end_position = bone->start_position;
bone->end_position += bone->rotation * bone->dir * bone->length;
//printf("%.3lf, %.3lf, %.3lf, %.3lf\n", posture.bone_rotations[bone->idx].x(), postu
visited[0] = true;
```

     然後進入 BFS 去 trace 整的骨架

```
BFS(bone->child, visited, posture);
```

     比較要提一下的是，我本來以為骨架的樹應該是一般的 tree 結構，所以認為 bone 下面應該會有很多個 child，所以以為不管是 child 還是 sibling 都應該是一個陣列，結果我的 BFS 一直都 trace 不完整個骨架，之後去翻了 code 才發現是這樣建的(下圖)



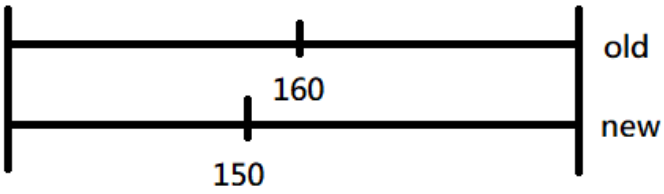
搞懂了 tree 的結構後，就改好 BFS。
Child 的 start position 會是 Parent 的 end position(這樣才接得起來)。

然後 rotation 的部分是 parent 的 rotation 加上自己與 parent 的 delta rotation(得到影片初始時候的位置)再加上當前 frame 的的 rotation 就能從 local space 轉到 world space 了。

至於 end position 就是上述算完後的角度在乘上該 bone 的長度

```
void BFS(acclaim::Bone* bone, bool visited[], const acclaim::Posture& posture)
{
    visited[bone->idx] = true;
    bone->start_position = bone->parent->end_position;

    bone->rotation = bone->parent->rotation * bone->rot_parent_current;
    bone->rotation *= util::rotateDegreeZYX(posture.bone_rotations[bone->idx]);

    bone->end_position = bone->start_position;
    bone->end_position += bone->rotation * (bone->dir.normalized() * bone->length);

    acclaim::Bone* temp = bone->sibling;
    while (temp != nullptr)
    {
        if ( !visited[temp->idx])
        {
            BFS(temp, visited, posture);
        }
        temp = temp->sibling;
    }
    if (bone->child != nullptr)
        if (!visited[bone->child->idx])
            BFS(bone->child, visited, posture);
}
```

- timeWarper:
  以 key frame 為中間點切為兩個部分



因為前半段比較少 frame(new)卻要跑比較多 frame(old)所以會加快，而後半段則相反

```
if (i <= keyframe_new)
{
  double interval = keyframe_old*1.0 / keyframe_new*1.0;
  double now_idx = interval * i;

  int Floor = now_idx;
  int Ceiling = now_idx + 1;

  double dist_to_p1 = now_idx - now_idx;

  Eigen::Quaterniond transform;

  transform.x() = postures[Floor].bone_translations[j].x();
  transform.y() = postures[Floor].bone_translations[j].y();
  transform.z() = postures[Floor].bone_translations[j].z();
  transform.w() = postures[Floor].bone_translations[j].w();

  Eigen::Quaterniond transform_next;
  transform_next.x() = postures[Ceiling].bone_translations[j].x();
  transform_next.y() = postures[Ceiling].bone_translations[j].y();
  transform_next.z() = postures[Ceiling].bone_translations[j].z();
  transform_next.w() = postures[Ceiling].bone_translations[j].w();

  Eigen::Quaterniond rotation;
  rotation.x() = postures[Floor].bone_rotations[j].x();
  rotation.y() = postures[Floor].bone_rotations[j].y();
  rotation.z() = postures[Floor].bone_rotations[j].z();
  rotation.w() = postures[Floor].bone_rotations[j].w();

  Eigen::Quaterniond rotation_next;
  rotation_next.x() = postures[Ceiling].bone_rotations[j].x();
  rotation_next.y() = postures[Ceiling].bone_rotations[j].y();
  rotation_next.z() = postures[Ceiling].bone_rotations[j].z();
  rotation_next.w() = postures[Ceiling].bone_rotations[j].w();

  Eigen::Quaterniond new_transform = transform.slerp(dist_to_p1, transform_next);
  Eigen::Quaterniond new_rotation = rotation.slerp(dist_to_p1, rotation_next);

  new_postures[i].bone_translations[j] = Eigen::Vector4d(new_transform.x(), new_transform.y(), new_transform.z(), new_transform.w());
  new_postures[i].bone_rotations[j] = Eigen::Vector4d(new_rotation.x(), new_rotation.y(), new_rotation.z(), new_rotation.w());
}
```

算出壓縮/拉長的比例是多少(interval)，然後得出現在的 frame 在 old frame 裡是第幾個 frame(now_idx)，然後拿前後的 frame 內插算出現在的 transform & rotation(因為算出來的 now_idx 不一定是整數，若非整數就是兩個 frame 中間的 frame，就要用內插求出來)

後半段也一樣，只有 interval 有差，進而影響 now_idx

```
double interval = (total_frames - keyframe_old) * 1.0 / (total_frames - keyframe_new) * 1.0;
double now_idx = interval * (i - keyframe_new) + keyframe_old -1;
```

4. Discussion / Conclusion

老實說這作業的觀念蠻簡單的，但是我還是寫了很久，主要的部分就是對 Eigen 這個 library 非常不熟，不知道該怎麼用，看了蠻多文檔其實還是不知道正規用法是怎麼用的，最後用強制轉型的方法實作看起來有點醜(?)，因為我蠻確定是哪些東西要乘、怎麼乘，所以基本上 70%以上的時間都在研究到底要怎麼乘起來(一直 type 有問題)，還有 20%在研究這整份 code 到底是怎麼實做的(不然不知道資料怎麼撈、結構長什麼樣子)

老實說我一直覺得電腦動畫的作業很好玩，只是因為我現在時間不夠，寫得很焦慮，不然有時候某些 BUG 還蠻好笑的😊