

Project 实验报告

一. 对于 parser 部分

1. 设计思路

首先从 sample.bvh 文件中读入所有数据，HIERARCHY 部分以字符串为单位存入队列，MOTION 里的 FRAME 部分直接读入 META 类的对象，每一帧的运动情况以二维 vector 形式储存。

然后利用函数递归建树，处理每一个结点时，从队列中弹出相应值赋值、取 vector 数组对应的运动参数赋值，遇到下一层结点进行递归，END 结点处理完后回退执行。

最后释放辅助内存，链接 META 和 joint 类变量，关闭文件。

2. 编程实现

(1) 读取 HIERARCHY 部分数据（队列 Q 定义为全局变量）

```
120     string s,t;                                //HIERARCHY
121     getline(bvhfile,s);
122     while(getline(bvhfile,s)){                  //按行读入HIERARCHY部分数据
123         if(!s.empty()){
124             s.erase(0,s.find_first_not_of(" "));
125             s.erase(s.find_last_not_of(" ") + 1);
126         }      //去掉每一行主体内容前、后的空格
127         if(s[0]=='M')break;
128         istringstream ss(s); //把每一行按空格分割为各子字符串存储
129         while(ss>>t)
130             Q.push(t);           //各子字符串按顺序入队
131     }
```

去除字符串两头的空格参考了：

<https://blog.csdn.net/qionggaobi9328/article/details/106123560>

提取字符串中由空格隔开的数据参考了：

<https://blog.csdn.net/longzaitianya1989/article/details/52909786>

(2) 读取 Frame 和 Frames 并赋值

```
133     getline(bvhfile,s);                      //Frame
134     istringstream f(s);    //处理第一行的Frames，能力有限，办法很简陋
135     f>>t; f>>t;          //就是先取出子字符串，将其格式转换成int类型，最后赋值
136     stringstream f_(t);
137     f_>>Frame;
138     meta_data.frame=Frame;
139     getline(bvhfile,s);
140     istringstream ft(s);
141     ft>>t; ft>>t; ft>>t;
142     stringstream ft_(t); //同理处理Frame Time
143     ft_>>meta_data.frame_time;
144     //cout<<meta_data.frame<< ' '<<meta_data.frame_time<<endl;
```

(3) 读取 MOTION 参数

```

147     while(getline(bvhfile,s)){           //MOTION
148         istringstream m(s);
149         vector<string> MM;
150         while(m>>t){      //把每一行的所有字符串数据插入辅助数组MM
151             MM.push_back(t);
152             //cout<<t<<' ';
153         }
154         M.push_back(MM);    //每一个子数组MM作为元素插入二维嵌套数组M
155         {
156             vector<string> tmp;
157             MM.swap(tmp);
158         }                      //释放辅助数组
159         //cout<<endl;
160     }

```

释放辅助数组的方法参考了：

<https://blog.csdn.net/sukhoi27smk/article/details/27505467>

之所以这么做是因为一开始遇到了一些内存方面的问题，就把每个 vector 都进行了类似处理，具体见问题与解决部分。

(4) 主体函数 CreTree () 部分

a. TYPE & NAME 很简单的赋值、出队

b. OFFSET 一样的道理完成结点的三个 offset 变量赋值

```

28     Q.pop();                                //OFFSET
29     stringstream x(Q.front()); //取队列当前的第一个字符串数据
30     x>>parent.offset_x;      //同样使用了stringstream，将字符串数据写入到double变量
31     //cout<<parent.offset_x<<' ';
32     Q.pop();                                //每完成一次赋值，相应的字符串数据出队

```

c. CHANNELS

```

42     Q.pop();                                //CHANNELS
43     int Channels;
44     stringstream ss(Q.front());
45     ss>>Channels;      //首先用Channels变量记录之后CHANNEL数据的个数
46     Q.pop();
47     string c;
48     for(int j=0;j<Channels;j++){
49         c=Q.front();          //利用中间变量c，按顺序完成对结点channels数组的赋值
50         parent.channels.push_back(c);
51         //cout<<c<<endl;
52         Q.pop();
53     }

```

d. MOTION

```

54     for(int i=0;i<Frame;i++){
55         vector<double> subm;           //MOTION
56         //subm.clear();
57         for(int j=col;j<col+Channels;j++){
58             double dm;
59             dm=stod(M[i][j]); //使用到了stod函数，可以将字符串转换成double类型
60             subm.push_back(dm);
61             //cout<<dm<<' ';
62         }
63         parent.motion.push_back(subm); //二维嵌套数组的每一个元素就是刚刚建好的子数组
64         //cout<<endl;
65     }
66     col+=Channels; //见文字说明

```

stod 函数的使用参考了网上说明：
https://blog.csdn.net/baidu_34884208/article/details/88342844

对于 col:

```

13     int Frame,col=0;
14     queue<string> Q;
15     vector<vector<string> > M;

```

motion 的每一帧参数都存储在二维嵌套数组 M 当中，每一行就是每一帧所有结点的所有运动参数。在读数据时，外循环 Frame 次，对应结点的 motion 数组长度为总帧数；而把 M 的每一行按列分割，就是按照 bvh 文件里每个人体关节出现的顺序，也就是建树时各结点创建的顺序，来依次分割成各结点的几个 channel 参数。所以定义了一个全局变量 col，初始值为 0。为每个结点的 motion 数组赋值时，就选取二维数组 M 中每一行的从 col 开始、数量为 Channels 的那几列，执行完毕后，col+=Channels，表示下一个结点的赋值从之后的列开始。但是这样处理一开始遇到了问题，具体见下面的问题与解决部分。

e. Children 部分 递归

```

67     string sub;                      //Children
68     sub=Q.front();
69     while(sub[0]!='}'){
70         joint current;
71         switch(sub[0]){
72             case('J'): { //如果下一结点是JOINT类型，就用递归建立子树current
73                 CreTree(current); //递归调用返回后，把子树存入parent的children数组
74                 parent.children.push_back(&current);
75                 break;
76             }
77             case('E'): { //如果下一结点是END类型的，要单独处理
78                 current.joint_type=Q.front();
79                 //cout<<current.joint_type<<' ';
80                 Q.pop();
81                 string n=parent.name;
82                 n.append("_End"); //利用字符串函数append()给子结点命名
83                 current.name=n;
84                 //cout<<current.name<<endl;
85                 Q.pop(); Q.pop(); Q.pop();
86                 stringstream x(Q.front()); //之后offset的处理同上
87                 x>>current.offset_x;

```

```

100     parent.children.push_back(&current);
101     Q.pop(); //处理完毕后，同样存入parent结点的children数组内
102     break;
103   }
104   default:;
105 }
106 sub=Q.front(); //继续取对头，进入下一轮循环，看parent是否有第二个子结点
107 }
108 Q.pop(); //处理完当前结点的所有子结点，出队，函数返回
109 return ;
110 }

```

(5) 释放 M 的内存，链接 META 和 joint 类的变量，关闭文件，程序运行结束

二. 对于 json 部分

1. 设计思路

有了 parser 部分的经验，这里实现起来就轻松了很多。首先打印出 meta_data 的两个值，然后进入 PriTree() 函数，按照 base 中建树的顺序遍历树的各个结点，并打印出每一个结点的 type、name、channels、motion 等值，最后关闭文件，结束程序。

2. 编程实现

(1) 主体函数 PriTree() 的实现和 base 中的 CreTree() 很类似，就是先打印出当前 parent 结点的 type、name、offset、channels 和 motion 的值，然后进入循环，遍历所有孩子结点，如果孩子结点是 JOINT 结点，就递归打印出子树；如果孩子结点是 End 类型的，特殊处理 motion 和 children 部分，其他同理，打印完成后要跳出循环并返回。

(2) 格式处理

- a. 使用 outfile<< 直接写入文件
- b. 空格字符串 blank 实现缩进

```

86     string blank="    ";
92     PriTree(&root,&blank);

```

我首先定义了一个内容为四个空格的字符串 blank，把它传入 PriTree() 函数

```

12 void PriTree(joint *parent,string *blank){
13   *blank+="    ";
14   outfile<<*blank<<'{'<<endl;
15   *blank+="    ";
16   outfile<<*blank<<"\"type\"<<": \"<<parent->joint_type<<"\",<<endl;

```

当进入内部层级，需要缩进四格时，就执行 *blank+=" "；

之后每一行的打印都先 outfile<<*blank 打印出前面的空格即可

```

75     outfile<<*blank<<"]"<<endl;
76     blank->erase(0,4);
77     outfile<<*blank<<"}"<<endl;
78     blank->erase(0,4);
79     return ;
80 }

```

当前结点打印完毕，函数返回之前，执行 blank->erase(0,4) 使得缩进回退

c. ,[]{} 与空行的处理

```
26 outfile<<*blank<<"\\"motion\\""<<": ["<<endl; //先输出motion部分的提示信息
27 *blank+=" ";
28 for(i=0;i<frame-1;i++){
29     outfile<<*blank<<"[";
30     for(j=0;j<c-1;j++)
31         outfile<<parent->motion[i][j]<<, ";//内循环c-1次，输出当前结点一帧下每个通道的值
32     outfile<<parent->motion[i][j]<<","<<endl; //除了最后一个通道，前面的输出后都要加上逗号
33 }
34 outfile<<*blank<<"[";
35 for(j=0;j<c-1;j++)
36     outfile<<parent->motion[i][j]<<, ";//最后一帧单独输出
37 outfile<<parent->motion[i][j]<<"]"<<endl; //最后一个通道输出后加上回车符，换行
38 blank->erase(0,4); //motion部分全部输出结束后，缩进要回退
39 outfile<<*blank<<","<<endl; //并输出整个motion的回车符，换行
```

上图以 motion 部分的处理为例，其他类同

三. 总结

本实验主要考察文件的读写、树的建立和遍历等，虽然最终将 base 和 bonus 的功能都实现了，但是仍存在办法简陋、代码冗余等问题。

- 首先是读文件部分

因为真的完全不了解这方面知识，查找资料也花费了很长时间。一开始使用了整行读取、去除前后空格、再按照空格分隔成子字符串的方式来读取的方法，这在读 MOTION 部分的数据还好，但是在 HIERARCHY 部分就不免多此一举。后来再查资料时发现了 file>> 这样可以直接按照字符串来读取的命令，因此在 json 部分吸取经验，使用了 outfile<< 这样的方式直接写，便捷很多。

- 还有就是 base 运行的时候终端的报错

```
terminate called after throwing an instance of 'std::length_error'
  what():  basic_string::_M_create
Aborted
```

一开始提示在_M_create 里有 length_error。检查后以为是 CreTree() 函数内的 col 变量值不对，导致用 M[i][j] 来访问数据的时候超出了最大长度，但是注释掉 col+=Channels 这一句之后还是有错

```
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
Aborted
```

这个内存错误困扰了我很久，查了很多资料都指向数据量过大、vector 内存等问题。我首先考虑是否是数据量的问题，就手动把 Frames 改成只有 3 个，跑了一遍发现问题并没有解决；然后我考虑是不是 vector 的问题，我先是按照网上的说明添加了一些释放 vector 内存的操作，发现没有什么用，于是我开始考虑问题出现的位置。我检查了 output.json 文件的内容，发现一开始的输出都很正常：

```
{ output.json > ...
1 {
2     "frame": 433,
3     "frame_time": 0.00833333,
4     "joint":
5     [
6         {
7             "type": "ROOT",
8             "name": "hip",
9             "offset": [76.5323, 59.3615, 165.373],
10            "channels": ["Xposition", "Yposition", "Zposition", "Zrotation", "Xrotation", "Yrotation",
11            "motion": [
12                [89.4173, 85.3829, -41.7785, -0.763319, -9.39531e-05, 0.0138756],
13                [87.4573, 82.9976, -49.8309, -2.49861, 2.77174, -1.20937],
14                [87.5149, 83.0759, -49.5964, -4.21552, 5.70183, -2.51723],
15                [87.5168, 83.2925, -49.0933, -5.61121, 8.30025, -3.69033],
16                [87.5144, 83.4594, -48.7759, -6.46084, 10.0843, -4.49314],
```

The screenshot shows a code editor interface with a JSON file named 'output.json'. The file contains several arrays of coordinates. A syntax error is highlighted at line 410, column 29, with the message 'Expected comma or closing bracket json(517) [行 410, 列 29]'. The status bar at the bottom indicates the file is 517 bytes long.

```
409
410 [ 79.0233, 76.442, 145.939, 0.454472, 5.29287, 1.69363],
      [78.8339, 76.229, 146.713]
```

但是到 410 行的时候突然输出错误了，提示需要一个逗号，但是这里既不是根结点 motion 的最后，也不是什么转折点，不明白为什么程序停在了这里，并且跑了几遍都停在这个地方。

我先是把 motion 部分全都注释，然后把每一步都打印出来，根据打印值发现建树的过程好像没什么问题，但我发现每一次执行完 `parent->children.push_back(current)` 之后，`children` 数组里面的每一个值都是一样的，而且存储地址一样，所以当新的元素被插入数组时，由于存储地址一样，就会改变原先数组中的内容，覆盖了前一个孩子结点的信息。解决办法是对每个 `current`，都使用 `new joint` 来分配新的存储空间。

```
59 | joint *current=new joint;
```

经过调整，我把这句命令加在了每个 while 执行的第一步，成功解决了这一问题。