一、Instancing

1.基本工具

ThreeJS中提供了两种缓冲区BufferAttribute和InstancingBufferAttribute，BufferAttribute缓冲区用于存放每个网格顶点的信息，InstancingBufferAttribute缓冲区用于存放每个对象的信息。

Instancing方式渲染多个相同对象时，GPU只从内存中取一次数据，减少了GPU的访存次数，所以速度较快，但是传统的GUP创建的对象基本上都是完全相同的，且一般不用于处理带动画的模型。

2.动画处理

gl\_Position = projectionMatrix \* modelViewMatrix\* matrix2 \* matrix1 \* vec4( position, 1.0 );

（1）动画与骨骼的关系

Time=time1+k\*(time2-time1);

Bones[i].position=k\*tracks[time1]+(1-k)\*tracks[time2];

（2）骨骼与点的关系

Matix1=weight[b1]\*bone[k1]+weight[b1]\*bone[k2]+weight[b1]\*bone[k3]+weight[b1]\*bone[k4];

3.贴图多样性

Texture=texture2D(text[type],UV);

4.颜色多样性

gl\_FragColor = vec4 (Texture.r+color[0],Texture.g+color[1],Texture.b+color[2],1.0);

5.将模型分区处理

if(vPosition.y<0.15)type\_part=0.0;//位于下半身  
else if(vPosition.y<0.59) type\_part=1.0;//位于上半身  
else type\_part=2.0;//位于头部

6.shader程序的优化

顶点着色器中的代码每一帧的每个点都会调用一次，因为执行次数非常多，所以这里代码的细微变化会对系统性能产生极大的影响。

shader程序中最好不要出现分支结构，但是有些分支结构对系统性能的影响不大，

二、PM

1.基本工具

PM是建立在QEM模型简化算法的基础之上的，是QEM模型简化算法的逆过程，QEM算法的核心是找出当前最优先的删除点,QEM算法的评价标准是曲度，弯曲程度大的地方后简化，弯曲程度小的地方先简化。

2.PM处理的流程

a将模型格式进行规范化

b使用含有插件的meshlab进行模型简化

c将PM所需信息分段

d在场景中还原模型

3.PM与LOD的结合

PM在场景中还原模型时，可以将中间计算结果存储下来用于LOD

4.PM与视点依赖

PM可以选定某个区域先简化或者最后简化

----视点依赖功能还未实现

精细的模型的处理

PM+LOD

简单相同部件的处理

实例化渲染

大规模人群的处理

细粒度化

提高资源的复用度

3D模型

模型网格

切分

纹理贴图

使用不同纹理贴图进行搭配组合

不同像素精度的多级处理

骨骼动画

预处理

设计了专门用于骨骼动画的数据存储格式

人群

资源复用->3D模型分级复用

第一级，所有对象都共用的数据 网格（UV）、骨骼

第二级，选择性使用的数据 纹理贴图、动画

人物多样性->参数化差异

坐标变换差异（变换矩阵从移动、放缩、旋转三个维度丰富多样性）

第一级，人物坐标变换

形态差异

通过放缩实现人物整体上的形态差异

位置差异

人物的移动和旋转

第一级，网格点坐标变换（通过骨骼来实现）

骨骼导致的形态差异

动画差异

动画播放速度

播放的动画类型

材质差异

纹理贴图的搭配组合

色调调整

色调对换

色调加强

多个小部件组合成的大型复杂模型

服务器端(轻量化处理)

模型分解->找出重用组件->外壳抽取

网络传输（提高传输速度）

优先传输先看到的部件,重用部件只传输一次

客户端（提高初始渲染速度，渲染质量）

重用部件使用实例化渲染

高精度模型处理（PM+LOD）

服务器端

将复杂模型简化，并记录简化过程中删除和修改的点以及操作的次序

网络传输

先传输基础模型，然后传输更新模型所需的数据

客户端

先将基础模型添加到场景中，然后不断更新模型

每更新到一定程度就记录下当前精度的模型数据

当彻底恢复成原模型后通过记录的数据实现LOD

提高初始加载速度

初始加载速度

PM处理

外壳提取

静态模型的轻量化处理

模型组件拆分

找出重用组件

外壳抽取

动态模型

细粒化调度机制流程

开始，轻量化预处理，找出重用部件，确定传输次序，完成

顶点着色器

一、输入的数据

模型顶点的信息：

顶点的原坐标

顶点的UV

顶点的对应骨骼编号

实例化对象的信息：

对象的变换矩阵

对象的动画播放速度和动画类型

对象的各部位贴图类型和色调

共用的unifrom类数据：

骨骼数据

时间

二、数据处理

判断顶点所在的区域

计算方法：

根据顶点在模型中的原坐标，判断顶点在头、上身、下身哪个部分

功能：

实现模型的分区，方便片元着色器的处理

实例化对象变换矩阵

计算方法：

将输入的对象变换矩阵信息解码为矩阵

功能：

用于单个实例化对象的放缩、移动、旋转

骨骼动画变换矩阵

计算方法：

找出当前点对应的骨骼编号

判断是否骨骼是否运动（静止骨骼的数据单独存放）

静止：根据骨骼编号找到对应的骨骼矩阵

运动：根据速度和时间计算帧序号

根据骨骼编号和帧序号找到对应的骨骼矩阵

功能：

控制动画的播放速度、种类

不同动画的搭配

通过骨骼放缩实现人物的形态差异

数据的存储和传输：

将静止骨骼的数据单独存放：

每帧变换矩阵都相同的骨骼，只存放一帧的数据

设计了专门用于传输骨骼的数据的浮点数类型：

新的16位浮点数类型所占空间比32位的float减少了一半。

将骨骼数据以纹理贴图的格式传入着色器。

1. 输出

顶点对应屏幕上的位置

向片元着色器输出

顶点所在的区域，

贴图类型，

色调

片元着色器

一输入

共用的unifrom类数据：

纹理贴图

顶点着色器输入的数据：

顶点所在的区域

贴图类型

色调

二数据处理

UV计算：由于人的纹理贴图左右对称，纹理贴图只传入了左半部分，要将UV都对应到左侧。

色调调整

三输出

对应像素的颜色

轻量化处理

大规模动态模型

方法：实例化渲染、参数化差异、模型资源分级复用

应用：听众人群

由多个小部件组合成的复合模型

方法：实例化渲染、外壳提取、视锥剔除

应用：会议厅模型（尤其是椅子、门和墙壁部分）

高精度模型处理

方法：PM、LOD、视锥剔除

应用：嘉宾模型

静态

轻量级渐进式会议场景

网络渐进式传输

轻量级web3D渲染

动态

轻量级渐进式会议场景

外壳提取

模型去重

光照烘焙

模型分解

骨骼动画数据的预处理

网络渐进式传输

将模型分解后传输

确定资源传输优先级

烘焙贴图的分级传输

结合了LOD的PM

轻量级web3D渲染

通过实例化渲染复原重复部件

模型资源的分级复用

人群间差异的参数化

使用shader实现多样性

---------------

结合了LOD的PM

首先传输compact mesh

传输stream of records

original mesh

------------------

骨骼动画数据的预处理

插值计算每帧的骨骼状态

（结合骨骼间的父子关系）计算骨骼的世界矩阵

使用新的浮点数格式

使用新的数据格式

----------------------------------------------

骨骼动画的处理

制作起始帧

制作末尾帧

计算过渡帧

--------------

实例化渲染

顶点着色器

输入

Attribute属性变量

顶点的原坐标

顶点的UV

顶点的骨骼索引

实例化对象的变换矩阵

实例化对象的动画播放速度

实例化对象的动画类型

实例化对象的贴图类型

实例化对象的色调

Uniform一致变量

模型视图矩阵

投影矩阵

时间

采样器

骨骼动画数据

输出

内建输出变量

顶点对应屏幕上的位置

Varying变量

顶点所在区域

贴图类型

色调

片元着色器

输入

Uniform一致变量

贴图个数

采样器

纹理贴图

Varying变量

。。。

输出

内建输出变量

片元颜色