Cocos2dx骨骼动画详解

一：骨骼动画介绍

游戏中人物的走动，跑动，攻击等动作是必不可少，实现它们的方法一般采用帧动画或者骨骼动画。

帧动画与骨骼动画的区别在于：帧动画的每一帧都是角色特定姿势的一个快照，动画的流畅性和平滑效果都取决于帧数的多少。而骨骼动画则是把角色的各部分身体部件图片绑定到一根根互相作用连接的“骨头”上，通过控制这些骨骼的位置、旋转方向和放大缩小而生成的动画。

它们需要的图片资源各不相同，如下分别是帧动画和骨骼动画所需的资源图：

骨骼动画比传统的逐帧动画要求更高的处理器性能，但同时它也具有更多的优势，比如：

●更少的美术资源：骨骼动画的资源是一块块小的角色部件（比如：头、手、胳膊、腰等等），美术再也不用提供每一帧完整的图片了，这无疑节省了资源大小，能为您节省出更多的人力物力更好的投入到游戏开发中去。

●更小的体积：帧动画需要提供每一帧图片。而骨骼动画只需要少量的图片资源，并把骨骼的动画数据保存在一个 json 文件里面（后文会提到），它所占用的空间非常小，并能为你的游戏提供独一无二的动画。

●更好的流畅性： 骨骼动画使用差值算法计算中间帧，这能让你的动画总是保持流畅的效果。

●装备附件： 图片绑定在骨骼上来实现动画。如果你需要可以方便的更换角色的装备满足不同的需求。甚至改变角色的样貌来达到动画重用的效果。

●不同动画可混合使用： 不同的骨骼动画可以被结合到一起。比如一个角色可以转动头部、射击并且同时也在走路。

●程序动画： 可以通过代码控制骨骼，比如可以实现跟随鼠标的射击，注视敌人，或者上坡时的身体前倾等效果。

骨骼动画是cocos2d-x动画在人物渲染方面的技术,分二个部分:用于绘制人物的外观呈现（被称为蒙皮或者mesh）和用于mesh进行动画(造型和关键帧）的一组分层的相互连接的骨骼.

骨骼动画的优势?

1 骨骼绑定可以让动画更精准，更真实自然，并可通过程序动态控制

2 动画各部分采用拼接方式，占用位图 / 内存资源少

3 骨骼显示对象与骨骼的逻辑分离，可在不影响动画播放的情况下动态更换

在你的应用中,Cocos2d-x提供了拥有2d骨骼动画的方式.构建骨骼动画过程开始可能有点复杂,但随后用起来却非常简单,而且有一些工具可以简化此过程.

当使用骨骼动画时，动画由一些相互连接的骨骼组成。影响一个骨骼将会影响其所有的子对象。通过每根骨头上不同的变换组合，你会得到骨骼的各种造型。

二：骨骼动画使用

我们在程序里面需要使用骨骼动画的时候，首先需要加载对应的头文件。

```
#include "cocostudio/CocoStudio.h"
using namespace cocostudio;
```

然后我们首先需要加载骨骼动画对应的三个文件，png、plist、xml或者ExportJson文件，加载之后，所需要的资源就在内存里面，加载代码如下所示：

//加载骨骼动画的文件

ArmatureDataManager::getInstance()->addArmatureFileInfo("armature/knight.png", "armature/knight.plist", “armature/knight.xml");

ArmatureDataManager::getInstance()->addArmatureFileInfo("armature/weapon.png", "armature/weapon.plist", “armature/weapon.xml");

ArmatureDataManager::getInstance()->addArmatureFileInfo("armature/robot.png", "armature/robot.plist", “armature/robot.xml");

ArmatureDataManager::getInstance()->addArmatureFileInfo("armature/cyborg.png", "armature/cyborg.plist", “armature/cyborg.xml");

ArmatureDataManager::getInstance()->addArmatureFileInfo("armature/Dragon.png", "armature/Dragon.plist", "armature/Dragon.xml");

ArmatureDataManager::getInstance()->addArmatureFileInfo("armature/Cowboy0.png", "armature/Cowboy0.plist", “armature/Cowboy.ExportJson");

加载之后，我们在一些方法里面就可以使用了，使用方式如下所示：

//创建动画对象，从动画列表中选择下标为0的进行播放。

```
auto  armature = Armature::create("Dragon");
armature->getAnimation()->playWithIndex(0);
armature->setPosition(Point(400,400));
armature->getAnimation()->setSpeedScale(0.5);
armature->setScale(0.6f);
addChild(armature);
```

当然，我们也可以根据动画的名字进行播放，如下所示

```
armature->getAnimation()->play(“Fire");
```

该代码和playWithIndex是一样的效果。当然其他的属性我们也可以根据这个进行设置。

如何设置事件回调机制

```
armature->getAnimation()->setMovementEventCallFunc(CC_CALLBACK_0(HelloWorld::animationEvent, this, std::placeholders::_1, std::placeholders::_2, std::placeholders::_3));
```

方法声名如下

```
void animationEvent(Armature *armature,MovementEventType movementType, const std::string& movementID);
```

实现如下：

```
void HelloWorld::animationEvent(Armature *armature, MovementEventType movementType, const std::string& movementID)
{
    if (movementType == LOOP_COMPLETE)
    {
        if (movementID == "Fire")
        {
            ActionInterval *actionToRight = MoveTo::create(2, Vec2(900- 50, 600));
            armature->stopAllActions();
            armature->runAction(Sequence::create(actionToRight, CallFuncN::create( CC_CALLBACK_1(HelloWorld::callback1, this)), nullptr));
            armature->getAnimation()->play("Walk");
        }
        else if (movementID == "FireMax")
        {
            ActionInterval *actionToLeft = MoveTo::create(2, Vec2(  50, 600));
            armature->stopAllActions();
            armature->runAction(Sequence::create(actionToLeft, CallFuncN::create( CC_CALLBACK_1(HelloWorld::callback2, this)), nullptr));
            armature->getAnimation()->play("Walk");
        }
    }
}

void HelloWorld::callback1(Node* sender)
{
    auto armature=(Armature*)sender;
   armature->runAction(ScaleTo::create(0.3f, 0.24f, 0.24f));
   armature->getAnimation()->play("FireMax", 10);
}
void HelloWorld::callback2(Node* sender)
{
```

```
  auto armature=(Armature*)sender;
  armature->runAction(ScaleTo::create(0.3f, -0.24f, 0.24f));
  armature->getAnimation()->play("Fire", 10);
}
```

如何获取所有的骨骼，并且在枪所在的那个骨骼上添加一个子弹。首先，每一个骨骼都有一个对应的图片节点，我们将其进行获取，并且进行坐标系转换就可以正确的添加了，如下代码所示：

```
  //获取骨骼所对应的节点，将其转换为世界坐标系，在该位置添加一个子弹精灵（前提是必须知道骨骼所对应的名称）
      Vec2 p = armature->getBone(“Layer126")->getDisplayRenderNode()->convertToWorldSpaceAR(Vec2(0, 0));

          auto bullet=Sprite::create("CloseNormal.png");
          bullet->setPosition(p.x + 60, p.y);

          bullet->stopAllActions();
          bullet->runAction(CCMoveBy::create(1.5f, Vec2(350, 0)));
          this->addChild(bullet);
```

三：骨骼动画换装的实现

骨骼动画的换装，在我们游戏开发中经常使用，下面我们详细的讲解一下换装的实现。

在此之前，我们必须知道需要换装的骨骼的名称。下面的方法可以获取骨骼动画的名称

```
          auto dic=armature->getBoneDic();
          auto mapKeyVec = dic.keys();
          for(auto key : mapKeyVec)
          {
                log("%s",key.c_str());
          }
```

只有知道了骨骼的名称我们才能进行操作。

```
    //创建皮肤
      Skin* skin = Skin::createWithSpriteFrameName("parts-tail.png");
      Skin* skin2 = Skin::createWithSpriteFrameName(“parts-head.png");
```

```
        //获取当前显示的皮肤的下标
          int index = armature->getBone("armR")->getDisplayManager()->getCurrentDisplayIndex();
          log(“index=%d",index);

          //添加皮肤，下标依次增长
          armature->getBone("armR")->addDisplay(skin, 1);
          armature->getBone("armR")->addDisplay(skin2, 2);

        //替换皮肤
          armature->getBone("armR")->changeDisplayWithIndex(1, true);
          int index2 = armature->getBone("armR")->getDisplayManager()->getCurrentDisplayIndex();
          log(“index=%d",index2);
```

四：Spine骨骼动画

Cocos2d-x程序中，使用Spine动画首先需要包含spine的相关头文件。

```
#include <spine/spine-cocos2dx.h>
#include "spine/spine.h"
using namespace spine;
```

其常用方法如下：

创建一个Spine动画对象，将动画文件和资源文件导入。

```
auto skeletonNode = new SkeletonAnimation("enemy.json", “enemy.atlas");
```

骨骼动画往往是不止一个动画的，例如：当人物需要行走时，就设置播放动画为行走；当要发动攻击时，就设置播放动画为攻击。下面方法可以设置当前播放动画，其中参数false表示不循环播放，true表示循环播放。

```
skeletonNode->setAnimation(0, "walk", true);
```

setAnimation方法只能播放一种动画，所以当要连续播放不同的动画时，需要使用addAnimation方法来实现，它可以一条一条的播放不同的动画。

```
    skeletonNode->addAnimation(0, "walk", true);
    skeletonNode->addAnimation(0, "attack", false);
```

对于一般情况下，动画的切换要求两个动画完全能衔接上，不然会出现跳跃感，这个对于美术来说要求很高，而Spine加了个动画混合的功能来解决这个问题。使得不要求两个动画能完全的衔接上，比如上面的walk和attack动画，就是衔接不上的，直接按上面的办法播放，会出现跳跃，但是加了混合后，看起来就很自然了。哪怕放慢10倍速度观察，也完美无缺。这个功能在序列帧动画时是无法实现的，也是最体现Spine价值的一个功能。

```
        skeletonNode->setMix("walk", "attack", 0.2f);
        skeletonNode->setMix("attack", "walk", 0.4f);
```

设置动画的播放快慢可通过设置它的timeScale值来实现。

```
        skeletonNode->timeScale = 0.6f;
```

设置是否显示骨骼通过设置debugBones，true表示显示，false表示隐藏。

```
        skeletonNode->debugBones = true;
```

例子：创建一个player行走和攻击的动画，并且循环播放。

```
      auto skeletonNode = new SkeletonAnimation("enemy.json",
"enemy.atlas");
      skeletonNode->setMix("walk", "attack", 0.2f);
      skeletonNode->setMix("attack", "walk", 0.4f);

      skeletonNode->setAnimation(0, "walk", false);
      skeletonNode->setAnimation(0, "attact", false);
      skeletonNode->addAnimation(0, "walk", false);
      skeletonNode->addAnimation(0, "attact", true);

      skeletonNode->debugBones = true;

      Size windowSize = Director::getInstance()->getWinSize();
      skeletonNode->setPosition(Point(windowSize.width / 2,
windowSize.height / 2));
addChild(skeletonNode);
```