
بسم الله الرحمن الرحيم

دانشگاه صنعتی اصفهان

مهندسی برق و کامپیوتر

بررسی سیستم گراف انیمیشن در موتور بازی سازی آنریل و پیاده‌سازی یک سیستم انیمیشن با استفاده از OpenGL

پایان‌نامه کارشناسی مهندسی کامپیوتر

نامی نذیری

استاد راهنما

دکتر مازیار پالهنگ

۱۴۰۱

دانشگاه صنعتی اصفهان

مهندسی برق و کامپیوتر

پایان‌نامه کارشناسی رشته مهندسی کامپیوتر آقای نامی نذیری

تحت عنوان

بررسی سیستم گراف انیمیشن در موتور بازی سازی آنریل و پیاده‌سازی یک سیستم انیمیشن با استفاده از OpenGL

در تاریخ ۱۳۹۵/۱۰/۲۰ توسط کمیته تخصصی زیر مورد بررسی و تصویب نهایی قرار گرفت:

۱- استاد راهنمای پایان‌نامه	دکتر مازیار پالهنگ
۲- استاد داور	دکتر داور اول
۳- استاد داور	دکتر داور دوم
سرپرست تحصیلات تکمیلی دانشکده	دکتر تحصیلات تکمیلی دانشکده

فهرست مطالب

عنـــوان **صفحه**

فهرست شکل‌ها

عنـــوان **صفحه**

فهرست جدول‌ها

عنـــوان **صفحه**

فهرست الگوریتم‌ها

چکیده

کلمات کلیدی

فصل اول
پیاده سازی

در این بخش ابتدای به ابزارها و کتابخانه‌هایی که در پیاده‌سازی استفاده شده‌اند، اشاره کرده و سپس به بررسی پیاده‌سازی سیستم پواینمایی می‌پردازیم.

۱-۱ ابزارها

۱-۱-۱ OpenGL

OpenGL یک واسط برنامه نویسی کاربردی [1] است که با فراهم کردن توابع مختلف به توسعه‌دهندگان امکان دستکاری گرافیک و تصاویر را می‌دهد. OpenGL یک کتابخانه‌ی رندرینگ است. یک "شئ" به خودی خود در OpenGL مفهومی ندارد و به صورت مجموعه‌ای از مثلث‌ها و حالات مختلف درنظر گرفته می‌شود. بنابراین وظیفه‌ی ما است که بدانیم چه شئ‌ای در کدام قسمت صفحه رندر شده است. این کتابخانه تنها وظیفه‌اش، کشیدن تصاویری که است که می‌خواهیم به تصویر کشیده‌شوند. در این صورت اگر می‌خواهیم تصویری را به‌روزرسانی کنیم و یا به عنوان مثال شئ‌ای را تحرک دهیم باید به OpenGL درخواست دهیم که صحنه را دوباره‌برای ما رندر کند. [۱]

به صورت کلی OpenGL را می‌توان یک ماشین حالت بزرگ درنظر گرفت. هر حالت شامل مجموعه‌ای از متغیرها است که نحوه‌ی عملکرد OpenGL را مشخص می‌کند. به مجموعه‌ی این حالت‌ها OpenGL context نیز می‌گویند.

[1] API

در واقع context را می‌توان یک شئ درنظر گرفت که کل OpenGL را دربر می‌گیرد. عموما تمامی تغییرات، روی context فعلی اعمال می‌شود و سپس رندر می‌شود. [۱][۲]

GLFW ۲-۱-۱

از آنجایی که به‌وجودآوردن یک پنجره‌ی جدید و همچنین context وابسته به نوع سیستم‌عامل است بنابراین نیازمند کتابخانه‌ای هستیم که بتواند این موارد را برای ما مدیریت کند. GLFW یک کتابخانه‌ی منبع باز و چندپلتفرمی برای OpenGL است که یک API ساده و مستقل از پلتفرم برای تولید پنجره‌ها، زمینه‌ها [۱] و سطوح، خواندن ورودی و مدیریت رویدادها [۲] را ارائه می‌کند. این کتابخانه از سیستم‌عامل‌های ویندوز ، مک و لینوکس و سیستم‌های مشابه یونیکس پشتیبانی می‌کند. [۳]

GLAD ۳-۱-۱

کتابخانه‌های گرافیکی مانند OpenGL وظیفه‌ی پیاده‌سازی توابع گرافیکی را ندارند بلکه می‌توان آن‌ها را مانند یک هدر در زبان برنامه‌نویسی C++ دانست که تعریف اولیه توابع را دارند. پیاده‌سازی این توابع در درایورهای GPU قرار دارند. دسترسی به این اشاره‌گرهای تابع به خودی خود سخت نیست ولی از آنجایی که این اشاره‌گرها وابسته به پلتفرم هستند بنابراین کار طاقت فرسایی است. وظیفه‌ی کتابخانه‌ی GLAD فراهم سازی و کنترل این اشاره‌گرهای تابع است. [۴]

GLM ٤-۱-۱

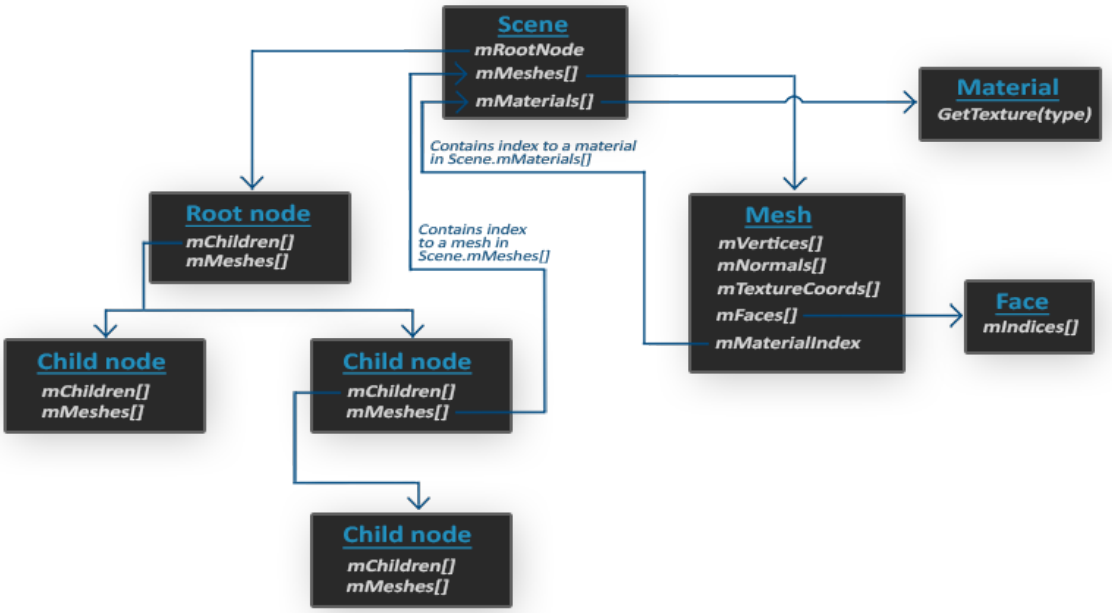
GLM یک کتابخانه‌ی ریاضی برای نرم‌افزارهای گرافیکی مبتنی بر زبان برنامه‌نویسی سایه‌ی OpenGL [۳] است. این کتابخانه تنها شامل یک هدر C++ است. توابع و کلاس‌های موجود در این کتابخانه به صورتی نامگذاری و طراحی شده‌آند که بسیار به GLSL نزدیک باشند.

Assimp ۵-۱-۱

Assimp یک کتابخانه برای بارگذاری و پردازش صحنه‌های هندسی از فرمت‌های مختلف است. می‌توان با استفاده از آن مواردی همچون مش‌های استاتیک و یا اسکلتونی، مواد [۴] ، انیمیشن های اسکلتونی و داده‌های بافت را از فایل بارگذاری کرد. زمانی که این مدل‌ها بارگذاری می‌شوند این کتابخانه آن‌ها را در ساختاری به شکل زیر ذخیره می‌کند

[1] Contexts
[2] Events
[3] OpenGL Shading Language(GLSL)
[4] Materials

و بعد از آن می‌توان از این ساختار، داده‌های مورد نظر خود را خواند و از آن‌ها استفاده کرد. [۵] [۶]



شکل ۱-۱ - ساختار کلاس‌های کتابخانه‌ی Assimp [۶]

۱-۱-۶ stb

این کتابخانه برای بارگذاری تصاویر استفاده می‌شود. در این پروژه از این کتابخانه برای بارگذاری تصاویر بافت‌ها در کنار کتابخانه‌ی Assimp استفاده شده است. [۷]

۱-۲ پیاده‌سازی سیستم پویانمایی

در این بخش به بررسی پیاده‌سازی انجام شده برای توسعه‌ی سیستم پویانمایی می‌پردازیم.

این پیاده‌سازی به چهار فاز مختلف تقسیم شده‌است. فاز اول به پیاده‌سازی محیط سه‌بعدی و نورپردازی می‌پردازد. در فاز بعدی به وارد کردن اشیاء سه‌بعدی می‌پردازیم. سپس به پیاده‌سازی سیستم پویانمایی و مواردی از جمله اسکل این بخش دو هدف کلی را دنبال می‌کند.

۱. نمایش مدل گرافیکی و اجرای انیمیشن بر روی آن

۲. ترکیب انیمیشن‌های مختلف به وسیله‌ی ماشین حالت

۱-۲-۱ نمایش مدل گرافیکی

همانطور که گفته شد مدل‌ها یا اشیاء سه‌بعدی به خودی خود مفهومی در OpenGL ندارند. آنچه برای OpenGL اهمیت دارد لیستی از مثلث‌ها است تا آن‌ها را به تصویر بکشد. مدل‌های سه‌بعدی از رئوس، لبه و وجوه تشکیل می‌شوند و در فرمت‌های مختلفی مانند FBX ذخیره می‌شوند. در این پیاده‌سازی، از کتابخانه‌ی Assimp برای خواندن این داده‌ها استفاده شده است.

۱-۲-۲ قرارگیری مدل سه‌بعدی در کارت گرافیک

آنچه برای OpenGL اهمیت دارد این است که به آن مجموعه‌ای از مثلث‌ها داده شود تا برایمان ترسیم کند. برای اینکار به صورت عمومی از ۳ آرایه مختلف استفاده می‌شود که به نام‌های VBO ، VAO و EBO شناخته می‌شوند. VBOs [۱] یک آرایه یا بافری است که تمامی رئوس مدل سه‌بعدی ما را در خود جای می‌دهد.

همانطور که در بخش ۲-۲-۱ اشاره شد، رئوس علاوه بر اینکه شامل اطلاعات موقعیت مکانی در محیط سه‌بعدی هستند، شامل اطلاعات دیگری نظیر رنگ، بردار نرمال، مختصات بافت و... نیز می‌توانند باشند. بنابراین باید به صورتی به کارت گرافیک اعلام کنیم که این داده‌ای که در آرایه‌ی VBOs قرار دارد را چگونه تفسیر کند. اینکار با استفاده از یک آرایه‌ی دیگر به نام VAO [۲] صورت می‌گیرد. در نهایت گفتیم که آنچه برای کارت گرافیک اهمیت دارد دریافت مثلث‌ها است. بنابراین باید به طریقی بگوییم کدارم رئوس با اتصال به یکدیگر مثلث تشکیل می‌دهند. اینکار نیز با استفاده از آرایه‌ی EBOs [۳] صورت می‌گیرد.

۱-۲-۳ سایه‌زنی فونگ

همانطور که در ؟؟ مطرح شد. الگوریتم سایه‌زنی فونگ شامل سه مولفه‌ی اصلی نور محیطی، نور پخش‌شده و نور آینه‌وار می‌شود. در نور محیطی تنها عامل تاثیرگذاری میزان قدرت منبع نور است. در نور پراکنده جهت قرار گیری منبع نور برای ما اهمیت پیدا می‌کند و در نهایت در نورپردازی آینه‌وار موقعیت بیننده به معادله اضافه می‌شود. هر کدام از این مولفه‌ها یک میزان روشنایی به ما داده و رنگ نهایی شئ به صورت زیر محاسبه می‌شود.

$$result = (ambient + diffuse + specular) * objectColor$$

[1] Vertex Buffer Objectss
[2] Vertex Array Objects
[3] Element Buffer Objects

نور محیطی

نور محیطی قدرت نورپردازی منبع نور را برای ما شبیه‌سازی می‌کند. برای افزودن نور محیطی به اشیاء موجود در صحنه‌ی سه‌بعدی بدین صورت عمل می‌کنیم، رنگ نور منبع نور را گرفته و آن را در یک ضریب محیطی ثابت کوچک ضرب کرده. این خروجی نورمحیطی است.

$$ambientStrength = 0.1$$

$$ambient = ambientStrength * lightColor$$

نور پخش‌شده

هرچه قطعات یک چندضلعی در جهت پرتوهای نور منبع نور باشند، نور پخش شده به آن قسمت روشنایی بیشتری می‌بخشد. بنابراین این مولفه تاثیر زاویه قرار گیری قطعات شئ با منبع نور را شبیه‌سازی می‌کند.

هر یک از قسمت‌های شی شامل یک بردار نرمال است. بردار نرمال برداری واحد و عمود بر شی است. با دانستن موقعیت مکانی منبع نور، می‌توان برداری از قطعه‌ی موجود در شی به منبع نور را بدست آورد. زاویه‌ی بین این بردار و بردار نرمال، زاویه‌ی بین قطعه‌ی شئ و منبع نور است. برای اینکه بفهمیم میزان تاثیرگذاری این نور بر روی آن قطعه چقدر است می‌توان از ضرب نقطه‌ای [1] این دو بردار استفاده کرد. سپس مقدار خروجی را در مقدار رنگ منبع نور کرده و این میزان نور پخش‌شده‌ی ما می‌شود. سلام

$$norm = normalize(Normal)$$

$$lightDir = normalize(lightPos - FragPos)$$

$$diff = max(dot(norm, lightDir), 0.0)$$

$$diffuse = diff * lightColor$$

نور آینه‌وار

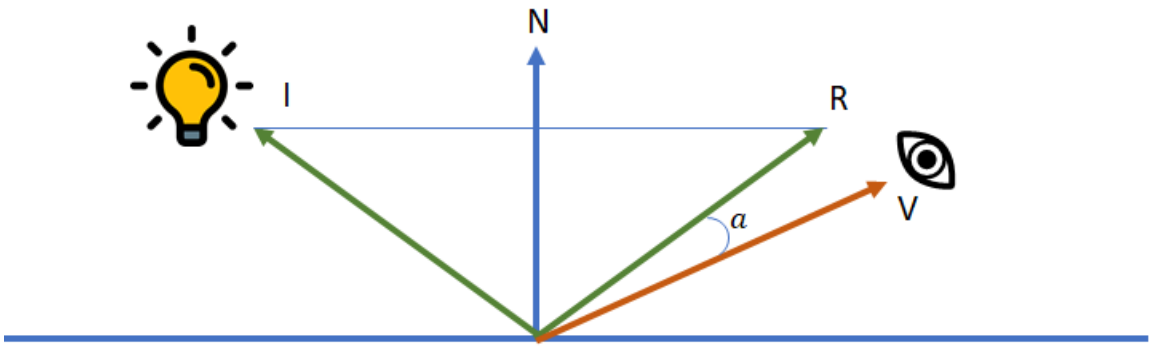
مانند نور پخش‌شده، نور آینه‌وار نیز بستگی به زاویه‌ی قرار گیری منبع نور و بردار نرمال قطعه دارد. تفاوت این مولفه این است که علاوه بر مورد اشاره شده وابسته به جهت مشاهده نیز دارد.

نورپردازی آینه‌وار بستگی بسیاری به خواص بازتابی سطوح دارد. اگر سطح جسم را مانند یک آینه درنظر بگیریم،

[1] Dot product

جایی را که بتوانیم نور منعکس شده را بر روی سطح ببینیم شدت آن نور قوی تر خواهد بود. بنابراین نتیجه‌ی نهایی این نورپردازی بدین صورت است که اجسام از زوایای خاصی روشن‌تر به نظر می‌رسند و هرچه از اجسام دورتر شویم از این روشنایی کاسته می‌شود.

برای بدست آوردن شدت این نور نیاز داریم ابتدا زاویه‌ی قرار گیری منبع نور با قسمت مورد نظر رو شئ را پیدا کنیم. این زاویه را در نور پخش‌شده نیز بدست آوردیم. سپس لازم است آن را نسبت به بردار نرمال شئ بازتاب دهیم. همچنین لازم است برداری بین بیننده و آن قطعه را نیز بدست آوریم. در نهایت زاویه‌ی بین بردار نور بازتاب داده‌شده و بردار موقعیت بیننده، زاویه‌ی مورد نظر ما است و برای بدست آوردن تاثیر شدت نور آینه‌وار می‌توان از ضرب نقطه‌ای این دو بردار استفاده کرد. از آنجایی که این نور بستگی به میزان براق بودن شئ دارد، بنابراین مانند فرمول زیر لازم است خروجی ضرب نقطه‌ای را به توان یک عددی که این عدد میزان براق بودن است برسانیم.



شکل ۱-۲ - نور آینه‌وار

$$specularStrength = 0.5$$

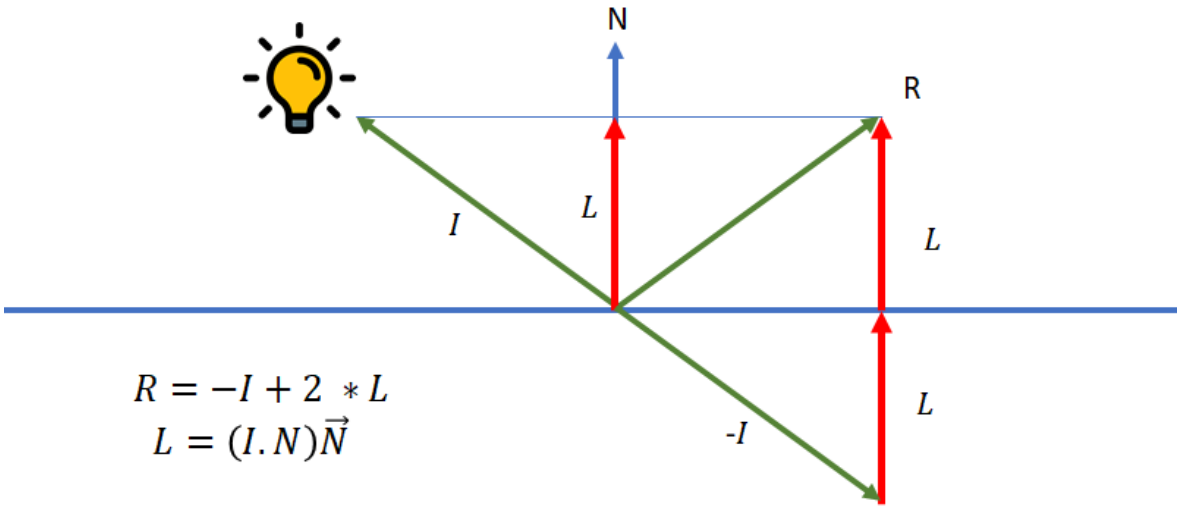
$$viewDir = normalize(viewPos - FragPos)$$

$$reflectDir = reflect(-lightDir, norm)$$

$$spec = pow(max(dot(viewDir, reflectDir), 0.0), 32)$$

$$specular = specularStrength * spec * lightColor$$

برای بدست آوردن بردار انعکاس می‌توان به صورت زیر عمل کرد.



شکل ۱-۳- بدست آوردن بردار پرتوی منعکس‌شده

۱-۲-٤ اسکلت شخصیت

اسکلت یک شخصیت به صورت مجموعه‌ای از مفاصل که به صورت سلسله مراتبی به یکدگیر متصل‌اند، تعریف می‌شود. در این پیاده‌سازی کلاس Bone نشان‌دهنده‌ی هر مفصل است. هر Bone یک والد دارد و می‌تواند به هر تعدادی فرزند داشته باشد. با توجه به تعریف آورده شده از اسکلت، کلاس اسکلت که با Skeleton مشخص شده، شامل لیستی از این مفاصل به همراه اشاره‌گری به مفصل ریشه است.

۱-۲-۵ اتصال اسکلت و مدل سه‌بعدی

اصطلاحی که برای اتصال اسکلت و مدل سه‌بعدی استفاده می‌شود Skinning است. در این روش هر راس موجود در مدل، به یک یا چند مفصل متصل می‌شود. الگوریتم به کاررفته در این پیاده‌سازی،الگوریتم linear blend skinning نام دارد. در این الگوریتم زمانی که یک راس به یک مفصل می‌شود به آن یک وزن نسبت داده می‌شود. این وزن نشان‌دهنده‌ی میزان تاثیرگذاری این مفصل بر روی این راس است. به بیانی دیگر، این وزن نشان می‌دهد که اکر این مفصل به مکان جدید منتقل شود، این انتقال چقدر بر روی آن راس تاثیر می‌گذارد. بنابراین برای بدست آوردن انتقال نهایی راس، باید انتقال راس را نسبت به هرکدام از مفاصلی که به آن متصل است را بدست آوریم، سپس انتقال

نهایی برابر مجموع وزن‌دار تمامی این انتقال‌ها خواهد بود.

۱-۲-۶ انیمیشن

هر بازی‌های کامپیوتری هر کلیپ انیمیشنی شامل یک حرکت منحصر به فرد شخصیت داخل بازی است. هر کلیپ شامل ژست‌های اسکلت در فاصله‌های زمانی مشخصی است. در واقع آنچه باعث حرکت شخصیت می‌شود حرکت اسکلت شخصیت است. زمانی که اسکلت شخصیت با استفاده از یک انیمیشن جابه‌جا می‌شود، مدل شخصیت نیز با استفاده از روش‌های skinning که در بالا توضیح داده‌شد همراه این اسکلت حرکت می‌کند.

انیمیشن‌ها از طریق کلاسی به اسم Animation Clip مدل‌سازی شده اند. این کلاس شامل آرایه‌ای از ژست های شخصیت در مدت زمان‌های مشخصی است. همراه یک اشاره‌گری به اسکلت شخصیت. نکته‌ی قابل توجه این است که هر کلیپ انیمیشنی مربوط به یک نوع اسکلت می‌شود. به زبانی دیگر نمی‌توان انیمیشنی که براس اسکلت شخصیت انسانی طراحی شده است را بر روی یک حیوان، مانند فیل اجرا کرد.

۱-۲-۷ پخش‌کننده‌ی انیمیشن

این سیستم وظیفه‌اش پخش کردن انیمیشن بر روی اسکلت شخصیت است. این سیستم با گرفتن یک انیمیشن و یک اسکلت، این انیمیشن را بر روی آن اسکلت اجرا می‌کند. همانطور که گفتیم، انیمیشن ها ژست شخصیت را در فاصله‌های زمانی مشخصی در خود ذخیره می‌کنند. وظیفه‌ی این سیستم این است که با استفاده از یک زمان‌سنج که نشان‌دهنده‌ی زمان فعلی بازی است ژست مناسب شخصیت را از داخل انیمیشن بدست آورد. قابل ذکر است که ممکن است این ژست با توجه به زمان بازی و فاصله‌های زمانی داخل انیمیشن از درون‌یابی دو ژست پشت سر هم در آن کلیپ بدست آید.

۱-۲-۸ الگوریتم پخش‌کننده‌ی انیمیشن

هر شخصیت درون بازی، اگر از نوع شخصیت اسکلتونی باشد، دارای یک پخش کننده ی انیمیشن خواهد بود.

در تصویر زیر تابع به‌روزرسانی اسکلت به وسیله‌ی انیمیشن را می‌توان مشاهده کرد.

```
    currentTime += deltaTime;

    const double currentAnimationTime = (currentTime - startTimeForCurrentAnim);
    AnimationPose currentPose = currentClip->GetPoseForCurrentFrame(currentAnimationTime *
        currentClip->GetFramePerSecond());

    SetSkeletonPose(currentPose);
```

برای اینکه بتوان یک انیمیشن را پخش کرد نیاز است دو مورد زیر را بدانیم.

۱. زمان فعلی درون بازی(CurrentTime)

۲. زمان شروع پخش انیمیشن فعلی(StartTimeForCurrentAnimation)

در ابتدا زمان فعلی درون بازی را برای این پخش‌کننده به‌روزرسانی می‌کنیم. سپس برای بدست آوردن زمان فعلی انیمیشن می‌توان از فرمول زیر استفاده کرد

CurrentAnimationTime = CurrentTime - StartTimeForCurrentAnimation

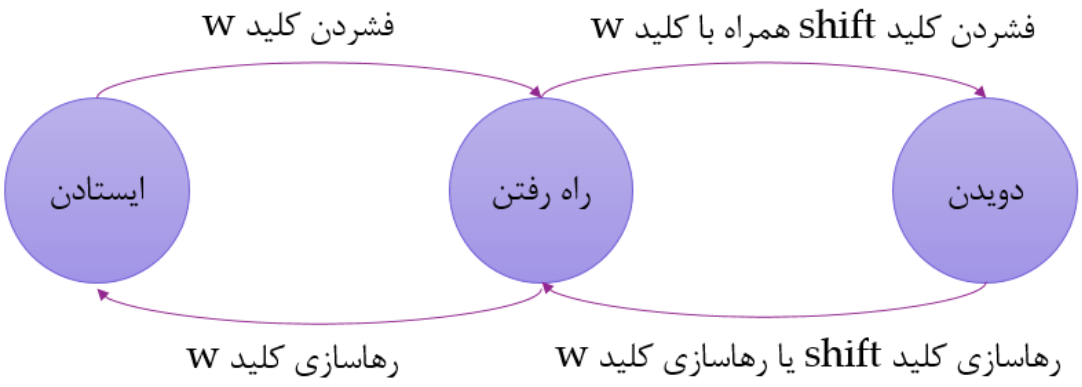
در نهایت با استفاده از این مقدار می‌توان ژست مورد نظر را از داخل کلیپ انیمیشنی بدست آورد. در نهایت نیز این ژست را بر روی اسکلت شخصیت اعمال می‌کنیم.

۱-۲-۹ ماشین حالت انیمیشن

یکی از روش‌های ترکیب انیمیشن‌های مختلف با یکدیگر، استفاده از ماشین حالت متناهی است. یک ماشین حالت متناهی شامل چندی حالت مختلف است که هر کدام از این حالات، حالتی از وضعیت سیستم را مشخص می‌کنند. زمانی که از ماشین حالت استفاده می شود سیستم می‌تواند در هر لحظه تنها در یکی از این حالات قرار گیرد. البته سیستم می‌تواند با دریافت ورودی از یک حالت به حالت دیگری رود.

دلیل استفاده از ماشین حالت متناهی برا سیستم انیمیشن این است که همانگونه که گفتیم، کلیپ‌های انیمیشنی، شامل ویدیوهای کوتاهی هستند که یک حالت مشخصی از شخصیت را بیان می‌کنند. در یک بازی، با توجه به ورودی بازیکن، شخصیت درون بازی می‌تواند در حالت‌های متفاوتی قرار گیرد. با استفاده از ماشین حالت می‌توان به تمامی این حالت‌ها رسیدگی کرد.

به عنوان مثال، تصویر زیر نشان‌دهنده‌ی یک ماشین‌حالت برای حرکت شخصیت است. شخصیت در ابتدا در حالت ایستاده قرار دارد و با گرفتن ورودی‌های مختلف از کیبورد، می‌تواند به حالت‌های دیگری رود.



شکل ۱-۴ - ماشین حالت برای حرکت شخصیت

برای پیاده‌سازی ماشین حالت متناهی، این سیستم به سه کلاس کلی شکسته‌شده‌است. کلاس AnimationStateMachine که وظیفه‌ی مدیریت حالت‌ها و انتقال از یک حالت به حالت دیگری را دارد. کلاس AnimationState که نشان‌دهنده‌ی حالت شخصیت است. هر AnimationState شامل یک کلیپ است و هر زمانی که این حالت فعال می‌شود این کلیپ پخش می‌شود. در نهایت کلاس Transition که شامل توابع انتقال است. هر حالت می‌تواند شامل چندین انتقال باشد. و وظیفه‌ی AnimationStateMachine است که بررسی کند، اگر انتقالی امکان‌پذیر بود، آن را انجام دهد.

۱-۲-۱۰ به‌روزرسانی ماشین حالت انیمیشن

وضعیت توابع انتقال تاثیرگذاری مستقیمی در وضعیت سیستم به‌روزرسانی ماشین حالت انیمیشن دارد. وضعیت انتقال می‌تواند سه حالت زیر را داشته باشد.

۱. حالت عادی [۱]

۲. حالت در حال انتقال [۲]

۳. حالت اتمام انتقال [۳]

حالت اول حالت عادی است که نشان‌دهنده‌ی وضعیت عادی ماشین حالت است. در این وضعیت، توابع انتقال حالت فعلی بررسی می‌شوند تا در صورتی که شرایطشان برقرار شود، تغییر حالت رخ دهد. علاوه بر آن انیمیشن حالت فعلی با استفاده از کلاس پخش‌کننده آپدیت می‌شود.

در صورتی که توابع انتقال مقدار درست [۴] را بازگردانند، ماشین به وضعیت دوم که وضعیت درحال انتقال است، تغییر وضعیت می‌دهد. در این وضعیت با توجه به زمانی که مشخص شده، ژست شخصیت با استفاده از درون‌یابی خطی از حالت فعلی به حالت جدید تغییر می‌کند.

پس از اینکه انتقال به صورت کامل انجام شد، وضعیت ماشین حالت به اتمام انتقال تغییر می‌یابد. زمانی که ماشین در این وضعیت قرار گرفته یعنی به حالت جدید منتقل شده، بنابراین لازم است انیمیشن را از حالت جدید گرفته و آن را به کلاس پخش کننده داده تا آن را پخش کند. پس از این کار وضعیت ماشین دوباره به حالت عادی تغییر می‌یابد و همه‌ی این موارد دوباره تکرار می‌شوند.

```
if(transitionStatus == TransitionStatus::normal)
{
  for (const auto& transition : currentState->GetTransitions()) // loop through
      transitions of the current state
  {
    if (transition->Evaluate())
```

[1] Normal
[2] Transitioning
[3] Finished
[4] True

```
        {
            transitionStatus = TransitionStatus::transitioning;
            currentState = animationStatesMap.at(transition->to);
            TransitionFromPose = animator->GetPoseAtCurrentTime();
            TransitionToPose = currentState->GetAnimClip()->GetPoseForCurrentFrame(0);
            currentTime = 0;
            transitionTime = transition->transitionTime;
            break;
        }
    }
}

if (transitionStatus == TransitionStatus::normal)
{
    animator->Update(deltaTime);
}
else if(transitionStatus == TransitionStatus::transitioning)
{
    if(TransitionUpdate(deltaTime))
    {
        transitionStatus = TransitionStatus::finished;
    }
}
else if(transitionStatus == TransitionStatus::finished)
{
    animator->ChangeAnimationClip(*(currentState->GetAnimClip()), 0);
    transitionStatus = TransitionStatus::normal;
}
```

پیوست‌ها

مراجع

[1] “Using opengl”, https://www.khronos.org/opengl/wiki/Getting_Started.

[2] “Opengl state macine”, https://learnopengl.com/Getting-started/OpenGL.

[3] “Glfw”, https://github.com/glfw/glfw.

[4] “Glad”, https://github.com/Dav1dde/glad.

[5] “Assimp”, https://assimp-docs.readthedocs.io/en/v5.1.0/about/introduction.html.

[6] “Assimp class hierarchy”, https://learnopengl.com/Model-Loading/Assimp.

[7] “stb”, https://github.com/nothings/stb.

Analysis of the animation graph in Unreal Engine and implementation of an animation system using OpenGL

Nami Naziri
nami.naziri@yahoo.com

May 22, 2022
Department of Electrical and Computer Engineering
Isfahan University of Technology, Isfahan 84156-83111, Iran
Degree: Bachelor of Science Language: Farsi

Supervisor: Maziar Palhang, Assoc. Prof., palhang@cc.iut.ac.ir.

Abstract

Keywords

Isfahan University of Technology

Department of Electrical and Computer Engineering

Analysis of the animation graph in Unreal Engine and implementation of an animation system using OpenGL

A Thesis

Submitted in partial fulfillment of the requirements

for the degree of Bachelor of Science

By

Nami Naziri

Evaluated and Approved by the Thesis Committee, on May 22, 2022

1- Maziar Palhang, Assoc. Prof. (Supervisor)

2- First Examiner, Assoc. Prof. (Examiner)

3- First Examiner, Assist. Prof. (Examiner)

Department Graduate Coordinator: Reza Tikani, Assist. Prof.