

لَا إِلَهَ إِلَّا اللَّهُ



دانشگاه صنعتی اصفهان  
مهندسی برق و کامپیوتر

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

پایان نامه کارشناسی مهندسی کامپیوتر

نامی نذیری

استاد راهنما  
دکتر مازیار پالهنک



دانشگاه صنعتی اصفهان  
مهندسی برق و کامپیوتر

پایان نامه کارشناسی رشته مهندسی کامپیوتر آقای نامی نذیری  
تحت عنوان

بررسی سیستم گراف انیمیشن در موتور بازی سازی آنریل و  
و پیاده سازی 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 پایان نامه دکتر مازیار پالهنک

۲- استاد داور دکتر داور اول

۳- استاد داور دکتر داور دوم

سرپرست تحصیلات تکمیلی دانشکده دکتر تحصیلات تکمیلی دانشکده

کلیه حقوق مالکیت مادی و معنوی مربوط به این پایان نامه متعلق به دانشگاه صنعتی اصفهان و پدیدآورندگان است. این حقوق توسط دانشگاه صنعتی اصفهان و بر اساس خط مشی مالکیت فکری این دانشگاه، ارزش گذاری و سهم بندی خواهد شد. هر گونه بهره برداری از محتوا، نتایج یا اقدام برای تجاری سازی دستاوردهای این پایان نامه تنها با مجوز کتبی دانشگاه صنعتی اصفهان امکان پذیر است.

## فهرست مطالب

عنوان	صفحه
فهرست مطالب.....	شش
فهرست شکل ها.....	نه
فهرست جدول ها.....	ده
فهرست الگوریتم ها.....	یازده
چکیده.....	۱
فصل اول: مقدمه	۲
فصل دوم: مرور	۵
۱-۲ تاریخچه ی پویانمایی سستی.....	۶
۱-۱-۲ فریم های کلیدی و درمیان.....	۶
۲-۱-۲ چشم انداز چند منظوره.....	۷
۳-۱-۲ لایه های مختلف.....	۷
۲-۲ پویانمایی کامپیوتری.....	۸
۱-۲-۲ فریم های کلیدی و درمیان.....	۹
۲-۲-۲ رویه.....	۹
۳-۲-۲ مبتنی بر فیزیک.....	۹
۴-۲-۲ ضبط حرکت <sup>۱</sup> .....	۹
۳-۲ موتور بازی سازی.....	۱۰
۴-۲ موتور بازی سازی آنریل.....	۱۰
۵-۲ زبان برنامه نویسی در آنریل.....	۱۱
۶-۲ گرافیک کامپیوتری.....	۱۱
۱-۶-۲ OpenGL.....	۱۲
۲-۶-۲ سایه زنی فونگ.....	۱۲
۳-۶-۲ نور محیطی.....	۱۲
۴-۶-۲ نور پخش شده.....	۱۳
۵-۶-۲ نور آینه وار.....	۱۳
۷-۲ مدل اسکلتونی.....	۱۳
۸-۲ شبکه ی <sup>۲</sup> چند ضلعی.....	۱۴
۱-۸-۲ راس.....	۱۴
۲-۸-۲ لبه.....	۱۴
۳-۸-۲ وجه.....	۱۴
۴-۸-۲ چند ضلعی.....	۱۵

<sup>1</sup>Motion Capture

<sup>2</sup>Mesh

۱۵.....	۹-۲ مدل
۱۵.....	۱۰-۲ زیرمش <sup>۱</sup>
۱۵.....	۱۱-۲ ماده <sup>۲</sup>
۱۶.....	۱۲-۲ بافت <sup>۳</sup>
۱۶.....	۱۳-۲ اسکلت
۱۶.....	۱۴-۲ Skinning
۱۶.....	۱۵-۲ ژست شخصیت
۱۷.....	۱۶-۲ کلیپ‌های انیمیشنی
۱۸.....	۱۷-۲ ترکیب انیمیشن
۱۹	فصل سوم: سیستم انیمیشن گراف در موتور بازی‌سازی آنریل
۱۹.....	۱-۳ بازیگران، پیاده‌ها و شخصیت‌ها
۲۰.....	۲-۳ اجزاء
۲۰.....	۳-۳ شخصیت‌ها
۲۱	فصل چهارم: معماری موتور بازی‌سازی آنریل
۲۱.....	۱-۴ UObject و Actors
۲۲.....	۲-۴ Components
۲۴	فصل پنجم: پیاده‌سازی
۲۴.....	۱-۵ ابزارها
۲۴.....	۱-۱-۵ OpenGL
۲۵.....	۲-۱-۵ GLFW
۲۵.....	۳-۱-۵ GLAD
۲۵.....	۴-۱-۵ GLM
۲۵.....	۵-۱-۵ Assimp
۲۶.....	۶-۱-۵ stb
۲۶.....	۲-۵ پیاده‌سازی
۲۶.....	۱-۲-۵ نمایش مدل گرافیکی
۲۷.....	۲-۲-۵ قرارگیری مدل سه‌بعدی در کارت گرافیک
۲۷.....	۳-۲-۵ اسکلت شخصیت
۲۷.....	۴-۲-۵ اتصال اسکلت و مدل سه‌بعدی
۲۸.....	۵-۲-۵ انیمیشن
۲۸.....	۶-۲-۵ پخش‌کننده‌ی انیمیشن
۲۸.....	۷-۲-۵ الگوریتم پخش‌کننده‌ی انیمیشن
۲۹.....	۸-۲-۵ ماشین حالت انیمیشن
۳۰.....	۹-۲-۵ به‌روزرسانی ماشین حالت انیمیشن

<sup>1</sup>Sub-Mesh

<sup>2</sup>Material

<sup>3</sup>Texture



## فهرست شکل ها

عنوان	صفحه
شکل ۱-۲: فریم های کلیدی و درمیان	۷
شکل ۲-۲: چشم انداز چند منظوره	۷
شکل ۳-۲: لایه های مختلف	۸
شکل ۴-۲: مولفه های الگوریتم سایه زنی فونگ [۱۱]	۱۳
شکل ۱-۴: تصویر UML کلاس های پایه ی موتور بازی سازی آنریل	۲۳
شکل ۱-۵: ساختار کلاس های کتابخانه ی Assimp [۱۷]	۲۶
شکل ۲-۵: ماشین حالت برای حرکت شخصیت	۲۹



## فهرست جدول‌ها

صفحه

عنوان



## فهرست الگوریتم‌ها

چکیده

کلمات کلیدی

## فصل اول

### مقدمه

پویانمایی هنر جان بخشیدن به اجسام بدون جان است. والت دیزنی درباره‌ی پویانمایی می‌گوید: ”پویانمایی می‌تواند هر آنچه را که ذهن انسان تصور می‌کند، توضیح دهد“

وقتی می‌گوییم جسمی را پویا کردیم، یعنی به آن جان بخشیدیم. زمانی که یک فیلم پویانمایی شده را در تلوزیون یا سینما می‌بینید، شخصیت‌های درون آن فیلم در حالت حرکت هستند. این حرکت معمولاً صاف و به هم پیوسته است. نوارهای حاوی فیلم متشکل از دنباله‌ای از تصاویر هستند که به عنوان ”فریم“ شناخته می‌شوند و درواقع با پخش شدن این فریم‌ها به صورت متوالی، توهم ایجاد حرکت به مخاطب ابراز می‌شود.

پویانمایی از گذشته تا امروز تغییرات فراوانی را دیده است. در پویانمایی سنتی، تصاویر به وسیله‌ی دست روی صفحات سلولوئیدی شفاف ترسیم یا نقاشی شده سپس از آن‌ها عکس گرفته و روی فیلم نمایش داده می‌شدند. امروزه اکثر پویانمایی‌ها با تصاویر کامپیوتری<sup>۱</sup> ساخته می‌شوند. [۱] علاوه بر این، دامنه‌ی استفاده از این پویانمایی نیز دستخوش بسیاری تغییرات شده است. در گذشته پویانمایی را می‌توانستیم در فیلم‌های پویانمایی شده یا کارتون‌ها مشاهده کنیم. اما اکنون با پیشرفت تکنولوژی، پویانمایی نقش بسیار اساسی‌ای در بازی‌های کامپیوتری پیدا کرده است. هدف بازی‌های کامپیوتری، به خصوص بازی‌های کامپیوتری داستان محور، غوطه‌ور کردن بازیکن در داستان است. همانطور که اشاره

---

<sup>۱</sup>CGI

شد پویانمایی هنر جان بخشیدن به اجسام است و به وسیله‌ی آن است که می‌توانیم احساسات و اعمال شخصیت بازی را به بازیکن منتقل کنیم.

بازی‌های کامپیوتری به صورت معمول توسط موتورهای بازی‌سازی ساخته می‌شوند. اگر بخواهیم تعریفی برای موتور بازی‌سازی آوریم می‌توان گفت آن‌ها پلتفرم‌هایی هستند که ساخت بازی‌های رایانه‌ای را آسان‌تر می‌کنند. موتورهای بازی‌سازی متشکل از مولفه‌های مختلفی هستند که قابلیت‌های لازم برای ساخت بازی را فراهم می‌کنند. از رایج‌ترین مولفه‌های موتور بازی می‌توان به مولفه‌ی صدا، مولفه‌ی رندر، مولفه‌ی هوش مصنوعی و مولفه‌ی انیمیشن اشاره کرد. [۲]

هدف اصلی این پروژه آشنایی با روش‌های استفاده شده در محیط‌های گرافیکی مانند موتورهای بازی‌سازی با تاکید بیشتر بر روی سیستم‌های پویانمایی به کار رفته در این محیط‌ها است.

به همین جهت این پروژه به دو صورت این هدف را دنبال می‌کند. جهت آشناشدن با یک موتور بازی‌سازی و نحوه‌ی پیاده‌سازی سیستم پویانمایی آن، موتور بازی‌سازی آنریل انتخاب شده است. آنریل یکی از معروف‌ترین موتورهای بازی‌سازی در جهان است که اولین نسل آن توسط تیم سوینی، بنیانگذار اپیک گیمز<sup>۱</sup>، توسعه یافت. آخرین نسخه‌ی این موتور به اسم موتور بازی‌سازی آنریل ۵ در سال ۲۰۲۰ معرفی و در سال ۲۰۲۲ انتشار یافت. سیستم پویانمایی این موتور بسیار وسیع است. به همین دلیل بخش کوچکی از این سیستم که گراف پویانمایی نام دارد، انتخاب شده و به بررسی ساختار و نحوه‌ی استفاده از این گراف می‌پردازیم.

پس از بدست آوردن تجربه‌ی اولیه از گراف انیمیشن برای آشنایی کامل تر با محیط گرافیکی و همچنین سیستم پویانمایی به پیاده‌سازی یک سیستم پویانمایی با استفاده از OpenGL پرداختیم. OpenGL یک واسط برنامه نویسی کاربردی<sup>۲</sup> است که با فراهم کردن توابع مختلف به توسعه‌دهندگان امکان دستکاری گرافیک و تصاویر را می‌دهد. با استفاده از این API می‌توان آشنایی خوبی در مورد گرافیک کامپیوتری و به صورت کلی محیط‌های گرافیکی بدست آورد. برای محیط سه‌بعدی پیاده‌سازی شده از روش Phong Shading برای نورپردازی محیط استفاده شده است. این روش یکی از معروف‌ترین روش‌های نورپردازی در محیط‌های سه‌بعدی بلادرنگ به‌خصوص بازی‌های کامپیوتری است. علاوه بر تولید صحنه‌ی سه‌بعدی، برای بدست آوردن آشنایی کامل با سیستم‌های پویانمایی که در بازی‌ها استفاده می‌شوند، به پیاده‌سازی یک نمونه از آن پرداختیم. در این پیاده‌سازی سیستم پویانمایی به چند بخش کلی تقسیم شده است که هر کدام هدف‌های مختلفی را دنبال می‌کند. برای اینکه یک سیستم پویانمایی داشته باشیم در ابتدا به یک شخصیتی نیاز داریم تا کلیپ‌های پویانمایی بر روی آن اجرا شود. شخصیت‌ها در این پیاده‌سازی توسط کتابخانه‌ی Assimp در ساختمان داده‌های مناسب ذخیره می‌شود. هر شخصیت در این پیاده‌سازی به دو قسمت کلی مش و اسکلت

<sup>۱</sup>Epic Games

<sup>۲</sup>API

تقسیم می‌شود. یکی از وظایف مهم این پیاده‌سازی، اتصال این دو قسمت به یکدیگر است. این اتصال به صورت کلی به اسم Skinning نام دارد. مرحله‌ی بعدی پیاده‌سازی به پخش کلیپ‌های پویانمایی بر روی این شخصیت می‌پردازد. در نهایت از ماشین حالت متناهی برای برای ترکیب کلیپ‌های پویانمایی متفاوت با یکدیگر استفاده شده است. خروجی این پروژه یک تحقیق در مورد سیستم گراف پویانمایی آنریل به همراه یک نرم‌افزار گرافیکی سیستم انیمیشن است.

در فصل‌های آتی به بررسی این موارد گفته‌شده پرداخته می‌شود. ابتدا در فصل دوم یک مروری بر تاریخچه‌ی انیمیشن‌ها می‌شود. سپس توضیحاتی درباره‌ی موتور بازی‌سازی و موتور بازی‌سازی آنریل داده می‌شود و در نهایت توضیحاتی کلی درباره‌ی روش‌های نورپردازی محیط و پویانمایی اسکلتونی که به وفور در موتورهای بازی‌سازی استفاده می‌شود داده می‌شود.

در فصل سوم به بررسی موتور بازی‌سازی آنریل با تاکید بر روی گراف پویانمایی می‌پردازیم و نحوه‌ی استفاده از آن را بررسی می‌کنیم.

در نهایت در فصل چهارم توضیحاتی درباره‌ی نحوه‌ی پیاده‌سازی سیستم پویانمایی به همراه توضیحات سیستم‌های موجود در این پیاده‌سازی می‌پردازیم.

در فصل "نتیجه‌گیری"، یک نتیجه‌گیری کلی از خروجی‌های این پروژه ارائه کرده و در فصل "کارهای آینده"، به بررسی مشکلاتی که می‌تواند در پیاده‌سازی برطرف شود به همراه پیشنهاداتی برای ادامه‌ی این پروژه پرداخته می‌شود. پویانمایی تاریخچه‌ای غنی دارد. در این فصل ابتدا به بررسی این تاریخچه با توضیحاتی درباره‌ی پویانمایی سنتی و پس از آن پویانمایی کامپیوتری پرداخته می‌شود. پس از آن به بررسی روش‌های کلی که توسط هنرمندان برای ایجاد پویانمایی به کار گرفته می‌شوند پرداخته می‌شود. در نهایت به بررسی به کارگرفتن این انیمیشن‌ها در موتورهای بازی سازی پرداخته می‌شود.

## فصل دوم

### مرور

از گذشته تا امروز نحوه‌ی تولید و همچنین استفاده از کلیپ‌های پویانمایی تغییرات زیادی کرده است. در گذشته اکثر پویانمایی‌ها با استفاده از دست و ترسیم بر روی صفحات سلولوئیدی تولید می‌شدند. در صورتیکه که امروزه، اکثر کلیپ‌های پویانمایی توسط تصاویر گرافیکی کامپیوتری ساخته می‌شوند. اگرچه با پیشرفت تکنولوژی روش‌های جدیدی برای تولید پویانمایی به وجود آمده، همچنان نیز از اکثر روش‌هایی که در پویانمایی سنتی استفاده می‌شد در پویانمایی کامپیوتری نیز استفاده می‌شود.

امروزه نه تنها از پویانمایی برای تولید فیلم و کارتون، استفاده می‌شود، بلکه آن‌ها نقش تاثیرگذاری در بازی‌های کامپیوتری پیدا کرده‌اند. برای تولید بازی‌های کامپیوتری از موتورهای بازی‌سازی استفاده می‌شود. موتورهای بازی‌سازی دارای مولفه‌های متفاوتی هستند که هر کدام وظیفه‌ی مخصوص به خودشان را دارند. مولفه‌ای که وظیفه‌ی پخش کلیپ‌های پویانمایی را دارد، مولفه‌ی پویانمایی گویند. بدیهی است اگر بخواهیم از سیستم پویانمایی استفاده کنیم نیازمند یک محیط گرافیکی هستیم. محیط‌های گرافیکی به صورت معمول از کارت گرافیک استفاده می‌کنند. برای دسترسی به توابع کارت گرافیکی می‌توان از یک رابط کاربری مانند DirectX یا OpenGL استفاده کرد. علاوه بر این برای مشاهده‌ی اشیاء موجود در محیط سه‌بعدی نیاز است که محیط را نورپردازی کنیم. روش‌های متنوعی برای نورپردازی محیط وجود دارند. یکی از این روش‌ها Phong Shading است که در این پروژه نیز استفاده شده است.



در اکثر سیستم‌های پویانمایی بازی‌های کامپیوتری از پویانمایی اسکلتی برای پخش و ترکیب کلیپ‌های پویانمایی استفاده می‌شود.

پویانمایی اسکلتی تکنیکی در پویانمایی کامپیوتری است که به وسیله‌ی آن، شخصیت‌های درون بازی متحرک می‌شوند. این سیستم به دو بخش کلی تقسیم می‌شود. یک بخش، یک مش یا پوسته است که برای به نمایش کشاندن شخصیت در محیط سه‌بعدی استفاده می‌شود و بخش دوم یک اسکلت است. این اسکلت مجموعه سلسله مراتبی از قطعات به هم پیوسته است که به هر قطعه یک مفصل گویند. در این تکنیک، اسکلت شخصیت درون بازی متحرک شده و با روش‌های موجود، پوسته یا مش آن شخصیت، اسکلت را دنبال می‌کند.

در این فصل، ابتدا مروری بر پویانمایی سنتی و روش‌های تولید آن می‌کنیم. سپس نگاهی به پیشرفت صنعت تولید پویانمایی کامپیوتری می‌اندازیم و روش‌های فعلی تولید پویانمایی را خواهیم گفت. پس از آن توضیحی در رابطه با موتورهای بازی‌سازی و به ویژه موتور آنریل خواهیم داد.

در قسمت انتهایی توضیحی در مورد Phong Shading که یکی از روش‌های نورپردازی محیط سه‌بعدی است خواهیم داد و علاوه بر آن در انتها توضیحات جامعی را در مورد پویانمایی اسکلتی ارائه خواهیم کرد

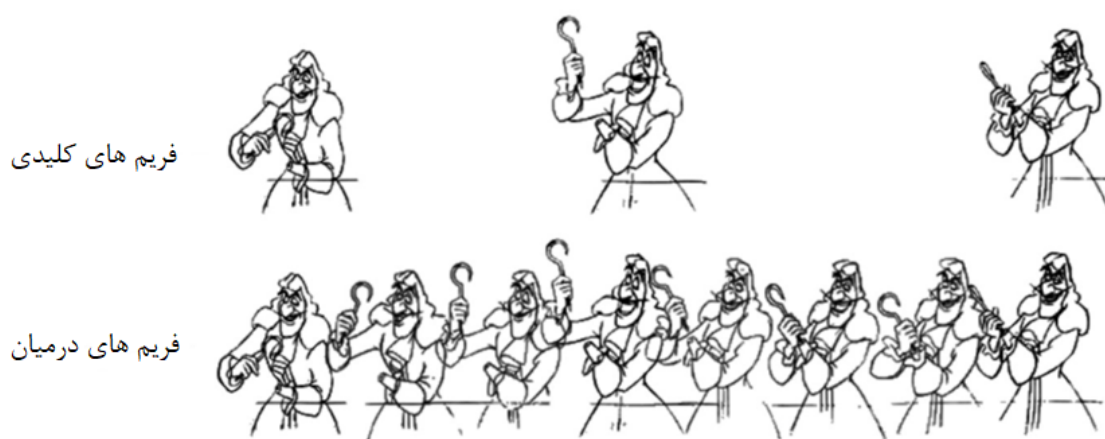
## ۲-۱ تاریخچه‌ی پویانمایی سنتی

پویانمایی سنتی که با اسم‌های مختلفی مانند "پویانمایی مرسوم"، "پویانمایی سل‌ای" و "پویانمایی بادست" شناخته می‌شود، روشی غالب برای تولید فیلم‌های پویانمایی شده در حدود قرن ۲۰ میلادی بود. در این روش، به صورت کلی پویانمایی به وسیله‌ی نقاشی با دست به وجود می‌آید. درواقع هر فریم از فیلم، یک عکس از نقاشی است و برای به وجود آوردن توهم حرکت، هر نقاشی اندکی با نقاشی قبلی خود تفاوت دارد.

برای تولید پویانمایی سنتی، از روش‌های مختلفی استفاده می‌شد. در اینجا به بررسی سه روش استفاده شده در پویانمایی سنتی می‌پردازیم.

### ۲-۱-۱ فریم‌های کلیدی و درمیان

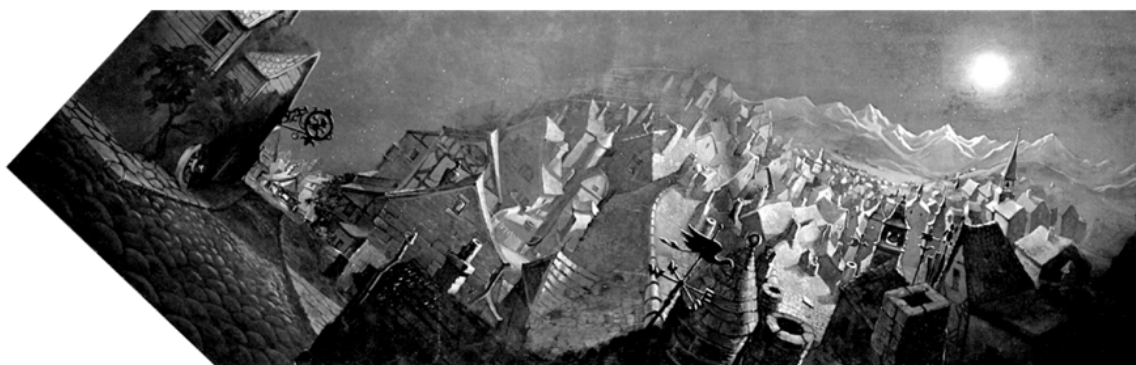
از آنجایی که تولید پویایی با دست و کشیدن نقاشی کار بسیار طولانی‌ای بود، برای اینکه وقت پویانمایی‌های ارشد ذخیره شود، این پویانماها فریم‌های اصلی یک حرکت را بر روی کاغذ ترسیم می‌کردند و فریم‌های میانی را پویانماهای جوان پر می‌کردند.



شکل ۱-۲ - فریم های کلیدی و درمیان

## ۲-۱-۲ چشم انداز چندمنظوره

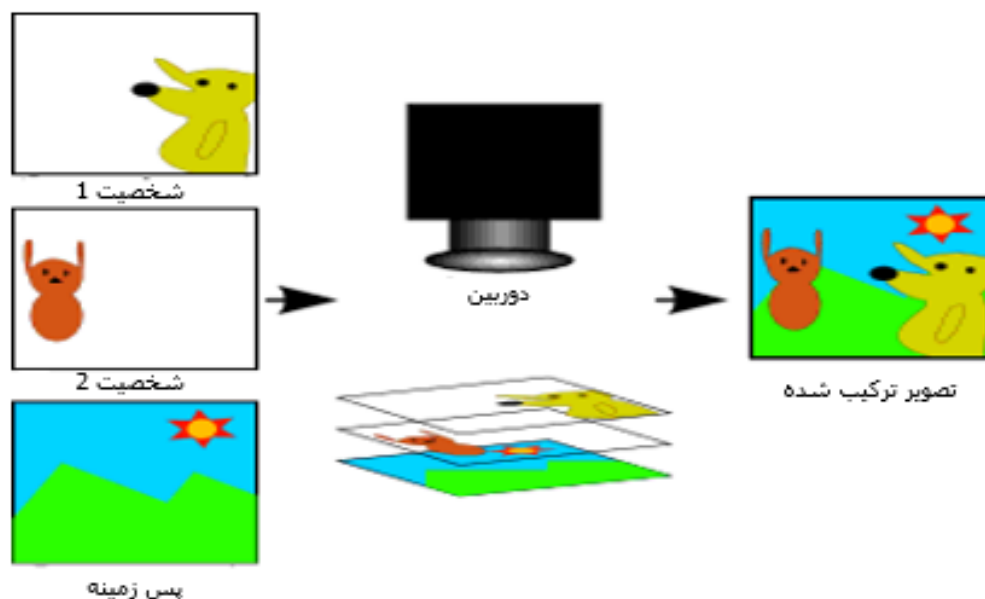
استفاده از چشم انداز چندمنظوره روش دیگری بود که در پویانمایی سنتی استفاده می شد. همطور که از تصویر زیر مشخص است، برای نمایش یک محیط از یک چشم انداز استفاده می شد. این چشم انداز می توانست نشان دهنده ی قرارگیری محیط در فواصل مختلف باشد. در این صورت، زمانی که دوربین در صحنه حرکت می کرد این توهّم را در مخاطب ایجاد می کرد که گویی در محیط در حال حرکت هستیم.



شکل ۲-۲ - چشم انداز چندمنظوره

## ۳-۱-۲ لایه های مختلف

با استفاده از این روش، پویانماها یک صحنه را به چند قسمت مختلف تقسیم می کردند. به عنوان مثال لایه های مختلفی برای هر شخصیت درون صحنه استفاده می شد. علاوه بر این یک لایه نیز برای تصویر پس زمینه استفاده می شد. از آنجایی که این لایه ها یک صفحه ی شفاف هستند بنابراین می توان لایه ها را بر روی هم انباشته کرد و با تصویر برداری از بالا تمام صحنه را تصویربرداری کرد. این روش در تصویر ۳-۲ آورده شده است.



شکل ۲-۳- لایه‌های مختلف

## ۲-۲ پویانمایی کامپیوتری

اگر بخواهیم نگاهی به تاریخچه‌ی انیمیشن‌های کامپیوتری بیاندازیم، مشاهده می‌کنیم که در حدود دهه‌ی ۱۹۸۰ میلادی شرکت دیزنی به عنوان یکی از اولین شرکت‌های جهان، شروع به دیجیتالی کردن خط لوله‌ی تولید پویانمایی سنتی خود کرد. در این دیجیتال‌سازی بسیاری از روش‌ها و ایده‌های استفاده شده در پویانمایی سنتی، به کار گرفته شد. اولین مقالات این حوزه توسط آقای جان لستر از کارمندان پیکسار به عنوان "اصول پویانمایی سنتی به کار رفته در پویانمایی کامپیوتری سه بعدی" ارائه شد. در این مقاله اصول اولیه پویانمایی سنتی دوبعدی ترسیم شده با دست و کاربرد آن‌ها در پویانمایی کامپیوتری سه بعدی شرح داده شده است. [۳]

پویانمایی کامپیوتری تنها محدود به دنیای سینما و فیلم‌های پویانمایی نمی‌شوند و به دنیای بازی‌های کامپیوتری نیز ورود پیدا کرده‌اند. بازی‌های کامپیوتری سعی می‌کنند دیوار میان تماشاگران و فیلم را بشکنند و با تعاملی بودن و دادن آزادی عمل به بازیکن، سعی می‌کنند داستان را به گونه‌ای تعریف کنند که گویی بازیکن یکی از شخصیت‌های اصلی داستان است. پویانمایی در بازی‌های کامپیوتری اهمیت بسیار بالایی دارد زیرا همانطور که گفته شد باعث جان بخشیدن به شخصیت‌ها می‌شود که اهمیت بسیار بالایی برای جلب توجه بازیکنان در هنگام داستان‌سرایی دارد.

با پیشرفت تکنولوژی، همراه با استفاده از روش‌های گذشته، روش‌های جدیدتری برای تولید پویانمایی توسعه یافته‌است که در ادامه به چند مورد از آن‌ها می‌پردازیم.

## ۱-۲-۲ فریم‌های کلیدی و درمیان

همانطور که اشاره شد در پویانمایی کامپیوتری از روش‌های موجود در پویانمایی سنتی استفاده شده است. در اینجا نیز فریم‌های کلیدی یک حرکت توسط پویانماها به وجود می‌آیند ولی فریم‌های میانی به جای اینکه توسط پویانماها به وجود آیند، توسط کامپیوتر با استفاده از روش‌های درونیابی به وجود می‌آیند.

### ۲-۲-۲ رویه

در این روش، حرکت بر اساس یک الگوریتم بیان می‌شود. درواقع انیمیشن‌ها در این نوع پویانمایی، تابعی با تعداد کمی از متغیرها هستند. به عنوان مثال یک تابعی را در نظر بگیرید که به گرفتن ورودی ثانیه، دقیقه و ساعت، یک شیء ساعت را خروجی دهد که عقربه‌هایش در جای مناسب با توجه به ورودی‌ها قرار گرفته باشد. حال می‌توان با تغییر ورودی‌ها حرکت ساعت را شبیه‌سازی کنیم.

### ۳-۲-۲ مبتنی بر فیزیک

پویانمایی مبتنی بر فیزیک پلی میان دنیای پویانمایی با دنیای واقعی است. در این روش با نسبت دادن ویژگی‌های فیزیک به اشیاء سه‌بعدی و سپس حل کردن فرمول‌های فیزیک مانند فرمول حرکت یا فرمول‌های نیوتن، فیزیک را شبیه سازی می‌کند. پویانمایی‌های مبتنی بر فیزیک شخصیت را قادر می‌سازد تا حرکت‌های خود را به صورت پویا با محیط تنظیم کند.

### ۴-۲-۲ ضبط حرکت<sup>۱</sup>

به فرآیند ثبت و دیجیتالی کردن حرکت یک شیء یا شخص، ضبط حرکت گویند. ضبط حرکت توسط دوربین‌های مادون قرمز که تعدادی زیادی از آن‌ها در صحنه‌ی ضبط قرار دارند، صورت می‌گیرد. این دوربین‌ها به صورت شبکه به یکدیگر متصل هستند و پس از کالیبره شدن، آماده‌ی استفاده هستند. این دوربین‌ها با استفاده از نشانگرهای سفیدی که بر روی لباس بازیگران ضبط حرکت قرار دارد، داده‌های مورد نیازشان را دریافت می‌کنند. قابل ذکر است این نشانگرها بازتابنده‌ی مادون قرمز هستند که توسط دوربین‌ها دریافت می‌شود. در نهایت پویانماها به پاکسازی و پردازش این داده‌ها پرداخته تا آن را برای استفاده‌ی شخصیت‌های سه بعدی آماده کنند.

<sup>1</sup> Motion Capture

## ۳-۲ موتور بازی سازی

موتورهای بازی سازی پلتفرم هایی هستند که ساخت بازی های رایانه ای را آسان تر می کنند. آن ها به شما این امکان را می دهند تا عناصر بازی مانند انیمیشن، تعامل با کاربر یا تشخیص برخورد میان اشیاء را در یک واحد ادغام و ترکیب کنید. [۲] زمانی که از اصطلاح موتور بازی سازی استفاده می کنیم منظورمان نرم افزارهای قابل توسعه ای هستند که می توانند پایه و اساس بسیاری از بازی های مختلف باشند. [۴] موتورهای بازی سازی متشکل از اجزای مختلفی هستند که قابلیت های لازم برای ساخت بازی را فراهم می کنند. رایج ترین اجزای موتور بازی عبارتند از: [۲]

- مولفه ی صدا: نقش اصلی این مولفه تولید جلوه های صوتی در بازی است.
- موتور رندر: وظیفه اصلی این مولفه تبدیل داده های ورودی به پیکسل ها، برای به تصویر کشاندن بر روی صفحه است.
- مولفه هوش مصنوعی: این مولفه مسئولیت ارائه ی تکنیک هایی برای تعریف قوانین رفتار شخصیت هایی را دارد که توسط بازیکنان کنترل نمی شوند.
- مولفه انیمیشن: نقش اصلی این مولفه اجرای انیمیشن های مختلف مانند حرکت است.
- مولفه شبکه: وظیفه اصلی این مولفه قادر ساختن بازی همزمان بازیکنان با یکدیگر، از طریق استفاده از دستگاه های متصل به اینترنت است.
- مولفه منطق یا مکانیک بازی: این مولفه قوانین حاکم بر دنیای مجازی، ویژگی های شخصیت های بازیکنان، هوش مصنوعی و اشیاء موجود در دنیای مجازی و همچنین وظایف و اهداف بازیکنان را تعریف می کند.
- ابزارهای نرم افزاری: وظیفه اصلی این ابزارها افزایش راندمان و سرعت تولید بازی با موتور بازی سازی است. آن ها توانایی اضافه کردن بسیاری از عناصر مختلف را به بازی ها، از انیمیشن و جلوه های صوتی گرفته تا الگوریتم های هوش مصنوعی، را فراهم می کنند.
- یکی از مهم ترین مولفه های موجود در هر موتور بازی، مولفه ی انیمیشن آن است. در این پروژه به بررسی سیستم انیمیشن گراف که وظیفه ی پخش انیمیشن شخصیت های سه بعدی را در موتور بازی آنریل دارد می پردازیم.

## ۴-۲ موتور بازی سازی آنریل

اولین نسل موتور بازی سازی آنریل توسط تیم سوینی، بنیانگذار اپیک گیمز<sup>۱</sup>، توسعه یافت. سوینی در سال ۱۹۹۵ شروع به نوشتن این موتور برای تولید بازی تیراندازی اول شخص به اسم غیر واقعی<sup>۲</sup> کرد. نسخه ی دوم موتور بازی سازی آنریل در سال ۲۰۰۲ منتشر شد.

<sup>۱</sup>Epic Games

<sup>۲</sup>Unreal

نسخه سوم نیز در سال ۲۰۰۴ پس از حدود ۱۸ ماه توسعه، منتشر شد. در این نسخه، معماری پایه‌ای موجود در نسخه‌ی اول مانند طراحی شی گرا، اسکریپت‌نویسی مبتنی بر داده و رویکرد نسبتاً مازولار نسبت به زیرسیستم‌ها وجود داشت. اما برخلاف نسخه دوم که از یک خط لوله با عملکرد ثابت<sup>۱</sup> استفاده می‌کرد، این نسخه به صورتی طراحی شده بود تا بتوان قسمت‌های سایه‌زنی سخت‌افزاری<sup>۲</sup> را برنامه‌نویسی کرد.

موتور بازی‌سازی آنریل ۴ در سال ۲۰۱۴ در کنفرانس توسعه‌دهندگان بازی<sup>۳</sup> منتشر شد. این نسخه با طرح کسب‌وکار اشتراکی برای توسعه‌دهندگان در دسترس قرار گرفت. این اشتراک به صورت ماهانه، با پرداخت ۱۹ دلار آمریکا به توسعه‌دهندگان این اجازه را می‌داد تا به نسخه‌ی کامل موتور، از جمله کد منبع C++ آن دسترسی پیدا کنند. البته در سال ۲۰۱۵ اپیک گیمز موتور بازی‌سازی آنریل را به صورت رایگان برای همگان منتشر ساخت.

آخرین نسخه آنریل به اسم موتور بازی‌سازی آنریل ۵ در سال ۲۰۲۰ معرفی شد. این نسخه از تمام سیستم‌های موجود از جمله کنسول‌های نسل بعدی پلی‌استیشن ۵<sup>۴</sup> و ایکس‌باکس سری X/S<sup>۵</sup> پشتیبانی می‌کند. کار بر روی این موتور حدود دو سال قبل از معرفی آن شروع شده بود. در سال ۲۰۲۱ نسخه‌ای از آن به صورت دسترسی اولیه منتشر شد. به طور رسمی در سال ۲۰۲۲ نسخه‌ی کامل این موتور برای توسعه‌دهندگان انتشار یافت. [۵]

## ۵-۲ زبان برنامه‌نویسی در آنریل

موتور بازی‌سازی آنریل از زبان برنامه‌نویسی C++ به همراه اسکریپت بصری به نام Blueprint استفاده می‌کند. Blueprint یک سیستم برنامه‌نویسی کامل گیمپلی مبتنی بر مفهوم استفاده از رابط‌های مبتنی بر گره برای ایجاد عناصر گیمپلی از درون ویرایشگر است. این سیستم بسیار منعطف و قدرتمند است زیرا این توانایی را در اختیار طراحان قرار می‌دهد تا از طیف گسترده‌ای از مفاهیم و ابزارها که عموماً فقط در دسترس برنامه‌نویسان هستند استفاده کنند. [۶]

## ۶-۲ گرافیک کامپیوتری

گرافیک کامپیوتری زیرشاخه‌ای از علوم کامپیوتر است که روش‌های ترکیب دیجیتالی و دستکاری محتوای بصری را مطالعه می‌کند. اغلب اوقات این اصطلاح به مطالعه‌ی کامپیوتر سه‌بعدی اشاره دارد. [۷] گرافیک کامپیوتری را می‌توان در موارد مختلفی مانند طراحی رابط کاربری، رندر اشیاء هندسی، انیمیشن و بسیاری موارد دیگر استفاده کرد. ابزارهای مختلفی برای پیاده‌سازی گرافیک کامپیوتری استفاده می‌شوند. یکی از این ابزارها OpenGL است. در این

<sup>۱</sup>fixed-function pipeline

<sup>۲</sup>shader hardware

<sup>۳</sup>GDC

<sup>۴</sup>PlayStation 5

<sup>۵</sup>Xbox Series X/S

پروژه برای ایجاد محیط گرافیکی از OpenGL استفاده شده است. برای اینکه بتوان شخصیت‌ها و محیط سه‌بعدی را به وضوح مشاهده کرد از یک الگوریتم نورپردازی به نام سایه‌زنی فونگ استفاده شده است. در این بخش توضیح کوتاهی درباره‌ی OpenGL و این سایه‌زنی آورده شده است.

## ۱-۶-۲ OpenGL

OpenGL یک واسط برنامه نویسی کاربردی<sup>۱</sup> است که با فراهم کردن توابع مختلف به توسعه‌دهندگان امکان دستکاری گرافیک و تصاویر را می‌دهد. OpenGL یک کتابخانه‌ی رندرینگ است. یک "شیء" به خودی خود در OpenGL مفهومی ندارد و به صورت مجموعه‌ای از مثلث‌ها و حالات مختلف در نظر گرفته می‌شود. بنابراین وظیفه‌ی ما است که بدانیم چه شیء‌ای در کدام قسمت صفحه رندر شده است. این کتابخانه تنها وظیفه‌اش، کشیدن تصاویری که است که می‌خواهیم به تصویر کشیده شوند. در این صورت اگر می‌خواهیم تصویری را به‌روزرسانی کنیم و یا به عنوان مثال شیء‌ای را تحرک دهیم باید به OpenGL درخواست دهیم که صحنه را دوباره برای ما رندر کند. [۸]

به صورت کلی OpenGL را می‌توان یک ماشین حالت بزرگ در نظر گرفت. هر حالت شامل مجموعه‌ای از متغیرها است که نحوه‌ی عملکرد OpenGL را مشخص می‌کند. به مجموعه‌ی این حالت‌ها OpenGL context نیز می‌گویند. در واقع context را می‌توان یک شیء در نظر گرفت که کل OpenGL را دربر می‌گیرد. عموماً تمامی تغییرات، روی context فعلی اعمال می‌شود و سپس رندر می‌شود. [۸] [۹]

## ۲-۶-۲ سایه‌زنی فونگ

نورپردازی در دنیای واقعی بسیار پیچیده است و به عوامل بسیار زیادی بستگی دارد که با توجه به قدرت محدود پردازش برای ما چنین امکانی وجود ندارد که آن را به صورت کامل تخمین بزنیم. بنابراین برای نورپردازی از تقریب واقعیت با استفاده از مدل‌های ساده‌شده‌ی فیزیکی استفاده می‌شود. یکی از این روش‌ها مدل نورپردازی فونگ نام دارد. این مدل بر اساس سه مولفه‌ی اصلی عمل می‌کند. این سه مولفه، نور محیطی<sup>۲</sup>، نور پخش شده<sup>۳</sup> و نور آینه‌وار<sup>۴</sup> نام دارند. [۱۰]

## ۳-۶-۲ نور محیطی

نور معمولاً از یک منبع نور منفرد ساطع نمی‌شود، بلکه از منابع نوری زیادی که در اطراف ما پراکنده شده‌اند، حتی زمانی که به صورت مستقیم قابل مشاهده نیستند، نشأت می‌گیرد. حتی زمانی که هوا تاریک است، معمولاً هنوز مقداری

<sup>۱</sup>API

<sup>۲</sup>Ambient light

<sup>۳</sup>Diffuse light

<sup>۴</sup>Specular light

نور در جایی در جهان وجود دارد. مانند ماه در هنگام شب. بنابراین اجسام تقریباً هرگز به صورت کامل تاریک نیستند. اگر بخواهیم چنین مولفه‌ای را به صورت واقعی مدل‌سازی کنیم، الگوریتمی بسیار پرهزینه خواهد بود. به همین جهت در مدل فونگ برای اینکه بتوانیم نور محیطی را در اجسام سه بعدی مشاهده کنیم از یک رنگ ثابت کوچک نور استفاده می‌کنیم و آن را به رنگ نهایی هر شیء اضافه می‌کنیم. در این صورت به نظر می‌رسد که همیشه مقداری نور پراکنده در محیط سه بعدی وجود دارد. [۱۰]

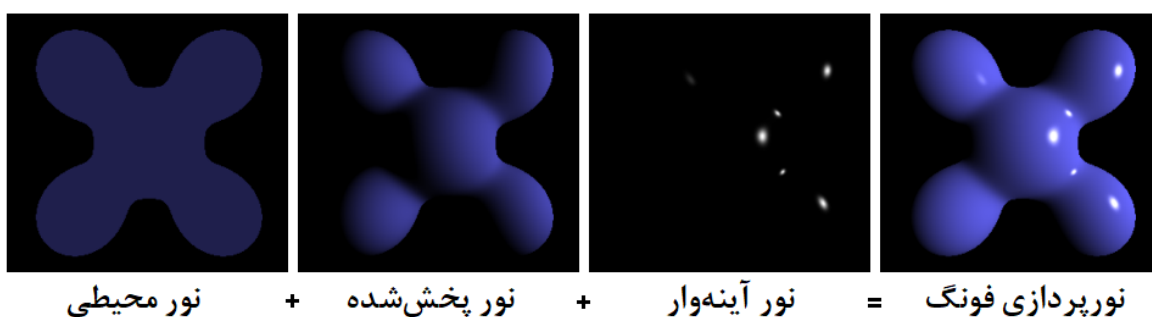
## ۲-۶-۴ نور پخش شده

می‌دانیم هرچه جسم به یک منبع نور نزدیک تر باشد و هرچه بخشی از یک جسم بیشتر به سمت منبع نور باشد، بیشتر روشن می‌شود. این مولفه، تاثیر جهت قرار گیری اجسام نسبت به منبع نور را شبیه‌سازی می‌کند. [۱۰]

## ۲-۶-۵ نور آینه‌وار

این مولفه وظیفه‌ی شبیه‌سازی نقطه‌ی روشن نوری را که بر روی اجسام براق ظاهر می‌شوند، دارد. جهت قرار گیری اجسام نسبت به جهت نور تاثیر زیادی بر نحوه‌ی شکل گیری و شمایل این نقطه‌ی روشن شده دارد. همچنین نورپردازی آینه‌وار تاثیرپذیری زیادی نسبت به جنس سطح و خواص بازتابی آن سطح دارد. هرچه سطح اجسام به جنس آینه‌ای نزدیک شود، نور بیشتری را بازتاب می‌دهد و بلعکس ممکن است جنس آن مانند گچی باشد که نور زیادی را جذب خود می‌کند. [۱۰]

در تصویر زیر می‌توانیم تمامی این مولفه‌ها را مشاهده کنیم.



نورپردازی فونگ = نور آینه‌وار + نور پخش شده + نور محیطی

شکل ۲-۴ - مولفه‌های الگوریتم سایه‌زنی فونگ [۱۱]

## ۲-۷ مدل اسکلتونی

در انیمیشن‌های اسکلتونی از مدل‌های اسکلتونی استفاده می‌شود. هر مدل اسکلتونی از دو بخش مدل و اسکلت تشکیل شده است.



## ۸-۲ شبکه‌ی<sup>۱</sup> چندضلعی

در گرافیک کامپیوتری سه‌بعدی و مدل‌سازی جامد، شبکه چندضلعی مجموعه‌ای از رئوس، لبه‌ها و وجوه است که شکل یک جسم چندوجهی را مشخص می‌کند. وجوه معمولاً از مثلث‌ها (شبکه مثلثی)، چهار ضلعی‌ها (چهار گوشه)، یا دیگر چند ضلعی‌های محدب ساده ( $n$  ضلعی‌ها) تشکیل شده‌اند. دلیل استفاده از این نوع چند ضلعی‌ها آسان‌تر بودن به نمایش کشیدن آن‌ها در محیط سه‌بعدی است. البته در حالت کلی اشیاء ممکن است از چندضلعی‌های مقعر و یا حتی چندضلعی‌های دارای سوراخ نیز تشکیل شده باشند.

اشیاء ایجاد شده توسط مش‌های چند ضلعی باید انواع مختلفی از عناصر، از جمله رئوس، لبه‌ها، وجوه، چندضلعی‌ها و سطوح را در خود ذخیره کنند. در بسیاری از نرم‌افزارهای سه‌بعدی، فقط رئوس، لبه‌ها و یکی از دو مورد وجوه یا چندضلعی‌ها ذخیره می‌شوند. در اکثر سیستم‌های رندر<sup>۲</sup> فقط از وجوه سه‌ضلعی (مثلث‌ها) استفاده می‌شود. بنابراین در این حالت چندضلعی‌های مدل، باید به شکل مثلث باشند. البته سیستم‌های رندر ای وجود دارند که از چهارضلعی‌ها یا چندضلعی‌های با تعداد اضلاع بالاتر نیز پشتیبانی می‌کنند و یا در لحظه این چندضلعی‌ها را به مجموعه‌ای از مثلث‌ها تبدیل می‌کنند که در این صورت باعث می‌شود نیازی به ذخیره‌ی مش به شکل مثلثی نباشد.

بنابراین چهار قسمت اصلی یک مش چندضلعی، رئوس، لبه‌ها، وجوه و چندضلعی‌ها هستند. توضیح کوتاهی درباره‌ی هر کدام از این موارد را در زیر می‌توانیم مشاهده کنیم.

### ۸-۲-۱ راس

راس‌ها معمولاً یک موقعیت در فضای سه‌بعدی همراه با اطلاعات دیگر مانند رنگ، بردار نرمال و مختصات بافت را شامل می‌شوند. در راس‌های مربوط به مش‌های اسکلتونی اطلاعاتی مانند تعداد مفاصلی که بر روی این راس تاثیر می‌گذارند همراه با وزن تاثیر گذاری آن‌ها، می‌تواند اضافه شود.

### ۸-۲-۲ لبه

ارتباط بین دو راس را لبه گویند.

### ۸-۲-۳ وجه

مجموعه‌ای بسته از لبه‌ها را وجه گویند. وجه‌ها می‌توانند از سه لبه (وجه مثلثی) یا از چهار لبه (وجه چهار گوش) تشکیل شده باشند.

<sup>1</sup>Mesh

<sup>2</sup>renderer

## ۲-۸-۴ چندضلعی

یک چندضلعی مجموعه‌ای همسطح از وجوه است. در سیستم‌هایی که از وجوه‌های چند ضلعی پشتیبانی می‌کنند، وجوه و چندضلعی‌ها یکسان هستند ولی در صورتی که سیستم مورد نظر تنها از سه یا چهار ضلعی‌ها پشتیبانی کند، در این صورت به چند ضلعی‌ها، مجموعه‌ای از وجوه گفته می‌شود.

## ۲-۹ مدل

مدل<sup>۱</sup> در واقع هر شی‌ای است که در محیط سه‌بعدی قرار می‌گیرد و به تصویر کشیده می‌شود. هر مدل می‌تواند از چند زیرمش تشکیل شود. به عنوان مثال یک ماشین را در نظر بگیریم. موجودیت ماشین می‌تواند یک مدل باشد که در محیط سه‌بعدی قرار می‌گیرد. مدل ماشین می‌تواند از چند زیرمش مانند چرخ‌ها، لاستیک‌ها و بدنه‌ی ماشین تشکیل شود. دلیل وجود داشتن یک موجودیت کلی به اسم ماشین این است که یک شخصی مانند طراح محیط و یا طراح مرحله نمی‌خواهد هر بار که ماشینی را در محیط قرار دهد، تک تک زیرمش‌ها را به صورت دستی در صحنه وارد کند و در سر جای خودش قرار بدهد.

## ۲-۱۰ زیرمش<sup>۲</sup>

چندضلعی‌های دارای یک نوع ماده<sup>۳</sup> را یک زیرمش گویند. همانطور که اشاره شد، هر مدل از چند زیرمش تشکیل می‌شود. دلیل این تقسیم این است که در هر عملیات به تصویر کشیدن<sup>۴</sup> تنها یک ماده می‌تواند به تصویر کشیده شود. مثلاً در مثال ماشین، قسمت‌های مختلف ماشین از ماده‌های مختلفی تشکیل می‌شود. به طور مثال چرخ ماشین می‌تواند از جنس آلومینیوم باشد، لاستیک چرخ از جنس پلاستیک باشد و یا حتی قسمت‌های داخلی ماشین مانند صندلی ماشین، از جنس چرم باشد. بنابراین باید این قسمت‌ها به صورت جدا قرار گیرند تا بتوان هر قسمت را با توجه به ماده‌ی مورد نظر آن به تصویر کشاند.

## ۲-۱۱ ماده<sup>۵</sup>

ماده‌ها شامل پارامترهای قابل تنظیمی هستند که با تنظیم آن‌ها، به گرافیک اعلام می‌شود که چگونه باید یک مثلث را به تصویر بکشد. این پارامترها می‌توانند شامل موارد زیر باشند ولی محدود به آن نمی‌شوند

### ۱. میزان کدورت و شفافیت شی

<sup>۱</sup> گاهی به جای استفاده از واژه‌ی مدل، از واژه‌ی مش هم استفاده می‌شود.

<sup>۲</sup> Sub-Mesh

<sup>۳</sup> Material

<sup>۴</sup> Render

<sup>۵</sup> Material

۲. میزان براقی شی

۳. رنگ (بافت) شی

۴. سایه زنی پیکسلی یا راسی<sup>۱</sup>

## ۱۲-۲ بافت<sup>۲</sup>

بافت یک تصویر دوبعدی و یا سه بعدی است که می تواند در ماده استفاده شود. این تصاویر به عنوان ورودی در برنامه دریافت شده و پس از اینکه یک شناسه به آن ها تخصیص داده شد، در کارت گرافیک قرار می گیرند. ماده ها با استفاده از این شناسه می توانند در صورت لزوم به این بافت ها دستیابی پیدا کنند.

## ۱۳-۲ اسکلت

به مجموعه ای از مفاصل که به صورت سلسله مراتبی به یکدیگر متصل می شوند، اسکلت گویند. پس از آنکه هنرمندان مدل شخصیت را طراحی می کنند در طی یک مرحله که به آن Rigging گویند، ساختار سلسله مراتبی اسکلت را به وجود می آورند. در انیمیشن ها در واقع این اسکلت است که حرکت می کند و با حرکتش باعث حرکت مدل شخصیت می شود.

## ۱۴-۲ Skinning

تا اینجا با دو مفهوم مدل و اسکلت آشنایی پیدا کردیم ولی نگفتیم که این دو چگونه به هم مرتبط می شوند. به عملیاتی که طی آن مفاصل موجود در اسکلت به مدل متصل می شود skinning گویند. طی این مرحله هر راس موجود در پوسته ی مش به یک یا چند مفصل متصل می شود. برای اینکه چگونه رئوس مش، این مفاصل را دنبال کنند الگوریتم های مختلفی مطرح شده است که در فصل پیاده سازی به یکی آن ها اشاره خواهد شد.

## ۱۵-۲ ژست شخصیت

ژست یک شخصیت نشان دهنده ی نحوه ی قرار گیری مفصل ها در اسکلت است. ژست های مختلف با دروان، حرکت یا تغییر اندازه ی مفاصل درون اسکلت به وجود می آیند. همانگونه که اشاره شد، اسکلت یک مدل در مرحله ی Rigging به وجود می آید و در همین مرحله با استفاده از Skinning به مدل متصل می شود. زمانی که این عمل صورت می گیرد

<sup>1</sup>Vertex or Pixel shader

<sup>2</sup>Texture

مدل در یک ژست به خصوص قرار دارد که به آن ژست حالت اتصال<sup>۱</sup> یا ژست مرجع<sup>۲</sup> گویند. به صورت کلی شخصیت در این حالت به صورتی ایستاده است که پاهایش کمی از هم باز است و بازوهایش به شکل حرف T کشیده است. به همین جهت گاهی به ژست حالت اتصال، ژست T<sup>۳</sup> هم گفته می شود. این حالت خاص به این دلیل انتخاب می شود که اندامها را از بدن دور نگه دارد و اینکار باعث می شود که فرایند اتصال رئوس به مفصل آسان تر شود. همانگونه که اشاره شد مفاصل به صورت سلسله مراتبی به یکدیگر متصل هستند. یعنی نحوه ی قرارگیری آنها متناسب با نحوه ی قرارگیری والدشان است. این کار باعث می شود که مفاصل به صورت طبیعی حرکت کنند. یعنی در صورتی که والد حرکت کند، به واسطه ی آن فرزند نیز حرکت می کند. زمانی که ژست شخصیت در این حالت والد، فرزندی قرار دارد به آن ژست محلی<sup>۴</sup> گفته می شود. حالت دیگری نیز وجود دارد که موقعیت هر مفصل نسبت به فضای مختصاتی مدل در نظر گرفته می شود. به ژست شخصیت در این حالت ژست جهانی<sup>۵</sup> گفته می شود.

## ۱۶-۲ کلیپ های انیمیشنی

در یک فیلم انیمیشنی، تمام بخش های یک صحنه قبل از ساخت هر انیمیشن به دقت برنامه ریزی می شود. این شامل حرکات هر شخصیت، لوازم موجود در صحنه و حتی حرکات دوربین نیز می شود. این بدان معنی است که کل صحنه را می توان به عنوان یک دنباله طولانی و پیوسته از فریم ها، متحرک ساخت. در این حالت در صورتی که شخصیتی خارج از دوربین هستند لازم نیستند که متحرک شوند.

کلیپ های انیمیشنی متفاوت از این هستند. یک بازی، یک تجربه ی تعاملی است بنابراین نمی توان از قبل چگونه حرکت کردن شخصیت ها و رفتار آنها را پیش بینی کرد. حتی تصمیمات شخصیت های غیربازیکن کامپیوتری نیز می توانند تابعی از اقدامات غیر قابل پیش بینی بازیکن انسانی باشد. به این ترتیب، کلیپ های انیمیشنی مربوط به بازی تقریباً هیچ گاه از مجموعه ای از فریم های طولانی و به هم پیوسته تشکیل نمی شوند. در عوض، حرکت شخصیت بازی باید به تعداد زیادی حرکات ریز تقسیم شود. منظور از کلیپ های انیمیشنی این حرکات کوتاه و یکتا است.

بنابراین هر کلیپ به صورتی طراحی شده است که یک عمل کاملاً مشخص را انجام دهد. برخی از این کلیپ ها به گونه ای طراحی شده اند که بتوان آن را به صورت حلقه شونده تکرار کرد. به عنوان مثال چرخه ی راه رفتن یا دویدن می توانند از این نوع کلیپ ها باشند. و حرکاتی مانند پریدن یا دست تکان دادن از نوعی هستند که تنها یک بار پخش می شوند.

بنابراین به طور کلی حرکات هر شخصیت بازی معمولاً به هزاران کلیپ تقسیم می شود. [۴]

<sup>1</sup>Bind Pose

<sup>2</sup>Reference Pose

<sup>3</sup>T Pose

<sup>4</sup>Local Pose

<sup>5</sup>Global Pose

## ۱۷-۲ ترکیب انیمیشن

اصطلاح ترکیب انیمیشن به هر تکنیکی اطلاق می‌شود که در آن بیش از یک کلیپ انیمیشن در ژست نهایی کاراکتر سهم می‌شود. به صورت دقیق تر در این عمل دو یا چند ژست برای ایجاد یک ژست خروجی برای اسکلت شخصیت، با یکدیگر ترکیب می‌شوند. همانطور که در بخش قبل گفته شد، کلیپ‌های انیمیشنی، کلیپ‌های کوتاه و یکتایی هستند. با استفاده از روش ترکیب می‌توان مجموعه‌ای از کلیپ‌های انیمیشنی را با یکدیگر ترکیب کرد تا مجموعه‌ی جدیدی از انیمیشن‌ها را بدون نیاز به ایجاد دستی و از پایه‌ی آن‌ها تولید کنیم.

به عنوان مثال، با ترکیب یک انیمیشن راه رفتن آسیب دیده با راه رفتن بدون آسیب دیدگی، می‌توانیم سطوح مختلفی از آسیب دیدگی در هنگام راه رفتن را به وجود آوریم. از ترکیب می‌توان برای درون‌یابی بین حالات مختلف چهره، حالت‌های مختلف بدن و حالت‌های مختلف حرکتی استفاده کرد. علاوه بر این می‌توان از آن برای یافتن یا حالت میانی بین دو حالت شناخته شده در زمان‌های مختلف نیز استفاده کرد. این کار زمانی استفاده می‌شود که بخواهیم ژست یک شخصیت را در نقطه‌ای از زمان پیدا کنیم که دقیقاً با یکی از فریم‌های نمونه موجود در داده‌های انیمیشن مطابقت ندارد. همچنین می‌توانیم از ترکیب موقتی انیمیشن برای انتقال هموار از یک انیمیشن به انیمیشن دیگر، با ترکیب تدریجی انیمیشن مبدا به مقصد در مدت زمان کوتاهی استفاده کنیم.

## فصل سوم

### سیستم انیمیشن گراف در موتور بازی سازی آنریل

در این فصل ابتدا توضیحاتی راجع به آنریل انجین داده می شود و سپس در رابطه ی سیستم انیمیشن گراف این انجین صحبت خواهد شد.

#### ۳-۱ بازیگران، پیاده ها و شخصیت ها

اشیا در آنریل می توانند به سه کلاس کلی بازیگران، پیاده ها و شخصیت ها دسته بندی می شوند. بازیگران کلاس پایه ی تمامی اشیا ای هستند که به صورت فیزیکی می توانند در محیط سه بعدی قرار گیرند. پیاده ها کلاسی مشتق شده از بازیگران هستند که بازیکنان می توانند کنترل آن ها را بدست گیرند و در محیط حرکت کنند. در نهایت شخصیت ها پیاده هایی هستند که دارای مش اسکلتونی، توانایی شناسایی برخورد و منطق حرکتی هستند. آنها مسئول تمام تعاملات فیزیکی بین بازیکن یا هوش مصنوعی، با جهان هستند و همچنین مدل های اولیه شبکه و دریافت ورودی را پیاده سازی می کنند. اگر بخواهیم شخصیت درون بازی از انیمیشن های اسکلتونی استفاده کند، باید از این کلاس بهره ببریم.

## ۲-۳ اجزاء

اجزاء<sup>۱</sup> مجموعه‌ای از توابع و ویژگی‌ها است که می‌تواند به یک بازیگر اضافه شود. بنابراین بازیگران می‌توانند حاوی مجموعه‌ای از ActorComponents باشند که این اجزاء می‌توانند برای موارد مختلفی از جمله کنترل نحوه‌ی حرکت بازیگران، نحوه‌ی رندر شدن و غیره استفاده شوند.

زمانی که یک مولفه به یک بازیگر اضافه می‌شود، آن بازیگر می‌تواند عملکردهای موجود در آن مولفه را استفاده کند. به عنوان مثال یک مولفه نور نقطه‌ای باعث می‌شود که بازیگر مانند یک نور نقطه‌ای، نور ساطع کند. یا یک مولفه صورتی به بازیگر این توانایی پخش صدا را می‌دهد.

مولفه‌ها حتماً باید به یک بازیگر متصل شوند و به خودی خود نمی‌توانند وجود داشته باشند. درواقع وقتی ما مولفه‌های مختلف را به بازیگر خود متصل می‌کنیم درواقع در حال قرار دادن قطعه‌ها و تکه‌هایی هستیم که مجموع آن‌ها یک بازیگر را به عنوان یک موجودیت واحد که در محیط سه‌بعدی قرار می‌گیرد تعریف می‌کنند. به عنوان مثال چرخ‌های یک ماشین، فرمان ماشین، چراغ‌ها و غیره همه به عنوان مولفه‌های ماشین در نظر گرفته می‌شوند در حالی که خود آن ماشین، بازیگر است.

## ۳-۳ شخصیت‌ها

هر شخصیت در آنریل از سه مولفه‌ی اصلی تشکیل شده است.

Skeletal Mesh Component □

Character Movement Component □

Capsule Component □

همانطور که در فصل‌های گذشته اشاره شد شخصیت‌ها برای پخش انیمیشن‌ها نیاز به یک مش اسکلتونی دارند. مولفه‌ی Skeletal mesh Component مش اسکلتونی اصلی مرتبط با شخصیت است. این مولفه‌ای است که برای ما در این پروژه اهمیت زیادی دارد.

مولفه‌ی Character Movement Component همانطور که از اسمش مشخص است برای منطق حرکت در حالت‌های مختلف از جمله راه‌رفتن افتادن و غیره استفاده می‌شود. این مولفه شامل تنظیمات و عملکردهای مربوطه برای کنترل حرکت است.

و در نهایت مولفه‌ی Capsule Component وظیفه‌ی تشخیص برخورد در هنگام حرکت را دارد.

---

<sup>1</sup>Components

## فصل چهارم

### معماری موتور بازی سازی آنریل

در این فصل ابتدا به معماری موتور آنریل پرداخته شده و پس از آن درباره ی ابزارهایی که این انجین در اختیار ما می گذارد صحبت می شود.

#### ۴-۱ UObject و Actors

کلاس پایه برای تمامی کلاس های دیگر در موتور آنریل UObject است. شیء ها<sup>۱</sup> نمونه هایی از کلاس هایی هستند که از UObject ها ارث می برند. بازیگران<sup>۲</sup> نمونه هایی از کلاس هایی هستند که از AActor ارث برده اند. کلاس AActor کلاس پایه برای تمامی اشیائی است که می توانند در جهان بازی قرار گیرند. به صورت کلی، بازیگران را می توان به عنوان یک کل یا موجودیت در نظر گرفت و اشیاء را قطعات تخصصی ای در نظر گرفت که در این موجودیت به کار می روند که به آن ها جزء<sup>۳</sup> می گویند. بنابراین اجزاء یک نوع خاصی از اشیاء هستند که بازیگران می توانند آن ها را به عنوان یک زیرشیء<sup>۴</sup> به خود متصل کنند.

به عنوان مثال اگر یک ماشین را در نظر بگیریم. ماشین به عنوان یک موجودیت کلی به عنوان بازیگر در نظر گرفته

---

<sup>1</sup> Objects

<sup>2</sup> Actors

<sup>3</sup> Component

<sup>4</sup> sub-object



می‌شود. در صورتی که قسمت‌های مختلف این ماشین مانند در ماشین یا چرخ ماشین اجزای آن ماشین در نظر گرفته می‌شوند. در ادامه این مثال، اگر کاربر قرار باشد که این ماشین را کنترل کند، یک جزء دیگر می‌تواند مسئولیت تغییر سرعت و جهت ماشین بر اساس ورودی کاربر را داشته باشد. [۱۲، ۱۳]

## ۲-۴ Components

همانطور که گفته شد، اجزاء نوع خاصی از اشیاء هستند که بازیگران می‌توانند به عنوان اشیاء فرعی به خود متصل کنند. کلاس پایه برای تمامی اجزاء، کلاس UActorComponent است. از آنجایی که استفاده از اجزاء تنها راه ممکن برای پرداخت<sup>۱</sup> مش‌ها<sup>۲</sup>، تصاویر، پخش صدا و در واقع هر چیزی که بازیکن هنگام بازی در جهان مشاهده یا تعامل می‌کنند هستند، بنابراین در نهایت از انواعی از این نوع اجزاء در توسعه‌ی بازی استفاده می‌شود.

برای ساخت اجزاء، چند کلاس اصلی وجود دارد که در هنگام ایجاد اجزاء باید به آن توجه کرد.

□ اجزای بازیگر<sup>۳</sup>: این کلاس بیشتر برای رفتارهای انتزاعی مانند حرکت، مدیریت موجودی یا ویژگی و سایر مفاهیم غیرفیزیکی مفید هستند. این نوع از اجزاء هیچ گونه مکان فیزیکی یا چرخشی در جهان ندارد.

□ اجزای صحنه<sup>۴</sup>: این کلاس فرزند کلاس اجزای بازیگر است و از رفتارهای مبتنی بر مکان پشتیبانی می‌کند که به نمایش هندسی نیاز ندارند. این کلاس می‌تواند شامل بازوهای فتری، دوربین‌ها، نیروهای فیزیک و حتی صدا شود.

□ اجزای اولیه<sup>۵</sup>: این کلاس فرزند کلاس اجزای صحنه است. در واقع این کلاس همان کلاس اجزای صحنه، همراه با نمایش هندسی است که عموماً برای نمایش عناصر بصری و برخورد<sup>۶</sup> یا همپوشانی<sup>۷</sup> با اشیاء فیزیکی استفاده می‌شود. این کلاس می‌تواند شامل مش‌های استاتیک<sup>۸</sup> یا اسکلتی<sup>۹</sup>، اسپرایت‌ها یا بیلبردها، سیستم‌های ذرات<sup>۱۰</sup> و همچنین حجم برخورد<sup>۱۱</sup> جعبه، کپسول و کره شود.

[۱۳]

<sup>۱</sup>Render

<sup>۲</sup>Meshes

<sup>۳</sup>ActorComponent

<sup>۴</sup>SceneComponent

<sup>۵</sup>Primitive Components

<sup>۶</sup>collide

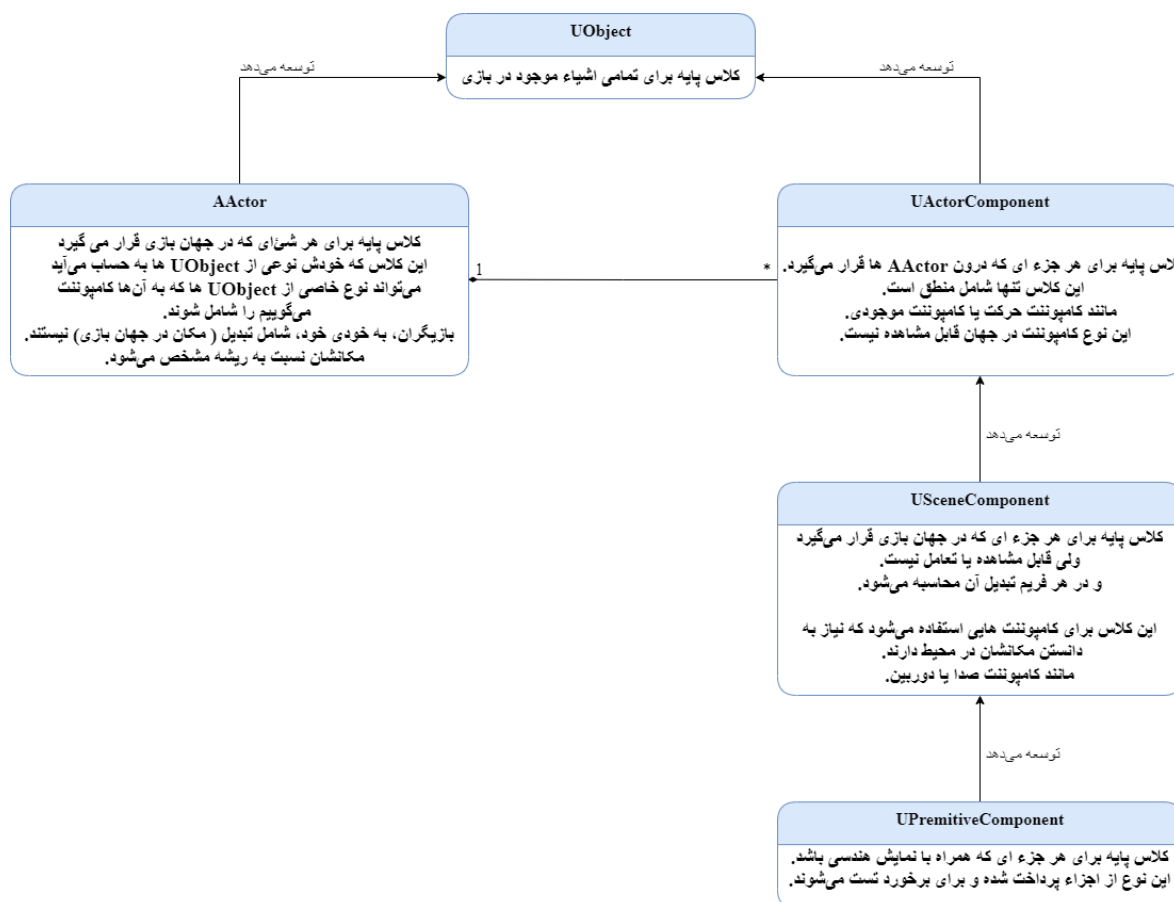
<sup>۷</sup>overlap

<sup>۸</sup>static mesh

<sup>۹</sup>skeletal mesh

<sup>۱۰</sup>particle systems

<sup>۱۱</sup>collision volumes



شکل ۴-۱ - تصویر UML کلاس های پایه ی موتور بازی سازی آنریل

## فصل پنجم

### پیاده سازی

در این بخش به روش ها و ابزارهای استفاده شده در پیاده سازی سیستم انیمیشن اشاره خواهد شد

#### ۱-۵ ابزارها

#### OpenGL ۱-۱-۵

OpenGL یک واسط برنامه نویسی کاربردی<sup>۱</sup> است که با فراهم کردن توابع مختلف به توسعه دهندگان امکان دستکاری گرافیک و تصاویر را می دهد. OpenGL یک کتابخانه ی رندرینگ است. یک "شیء" به خودی خود در OpenGL مفهومی ندارد و به صورت مجموعه ای از مثلث ها و حالات مختلف در نظر گرفته می شود. بنابراین وظیفه ی ما است که بدانیم چه شیء ای در کدام قسمت صفحه رندر شده است. این کتابخانه تنها وظیفه اش، کشیدن تصاویری که است که می خواهیم به تصویر کشیده شوند. در این صورت اگر می خواهیم تصویری را به روزرسانی کنیم و یا به عنوان مثال شیء ای را تحرک دهیم باید به OpenGL درخواست دهیم که صحنه را دوباره برای ما رندر کند. [۸]

به صورت کلی OpenGL را می توان یک ماشین حالت بزرگ در نظر گرفت. هر حالت شامل مجموعه ای از متغیرها است که نحوه ی عملکرد OpenGL را مشخص می کند. به مجموعه ی این حالت ها OpenGL context نیز می گویند. در واقع context را می توان یک شیء در نظر گرفت که کل OpenGL را دربر می گیرد. عموماً تمامی تغییرات، روی

---

<sup>۱</sup>API

context فعلی اعمال می شود و سپس رندر می شود. [۸] [۹]

### GLFW ۲-۱-۵

از آنجایی که به وجود آوردن یک پنجره‌ی جدید و همچنین context وابسته به نوع سیستم عامل 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<sup>۱</sup> و سطوح، خواندن ورودی و مدیریت رویدادها<sup>۲</sup> را ارائه می کند. این کتابخانه از سیستم عامل‌های ویندوز، مک و لینوکس و سیستم‌های مشابه یونیکس پشتیبانی می کند. [۱۴]

### GLAD ۳-۱-۵

کتابخانه‌های گرافیکی مانند OpenGL وظیفه‌ی پیاده‌سازی توابع گرافیکی را ندارند بلکه می توان 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 اشاره گرهای تابع به خودی خود سخت نیست ولی از آنجایی که این اشاره گر ها وابسته به پلتفرم هستند بنابراین کار طاقت فرسایی است. وظیفه‌ی کتابخانه‌ی GLAD فراهم سازی و کنترل این اشاره گرهای تابع است. [۱۵]

### GLM ۴-۱-۵

GLM یک کتابخانه‌ی ریاضی برای نرم افزارهای گرافیکی مبتنی بر زبان برنامه‌نویسی سایه‌ی OpenGL<sup>۳</sup> است. این کتابخانه تنها شامل یک هدر C++ است. توابع و کلاس‌های موجود در این کتابخانه به صورتی نامگذاری و طراحی شده‌اند که بسیار به GLSL نزدیک باشند.

### Assimp ۵-۱-۵

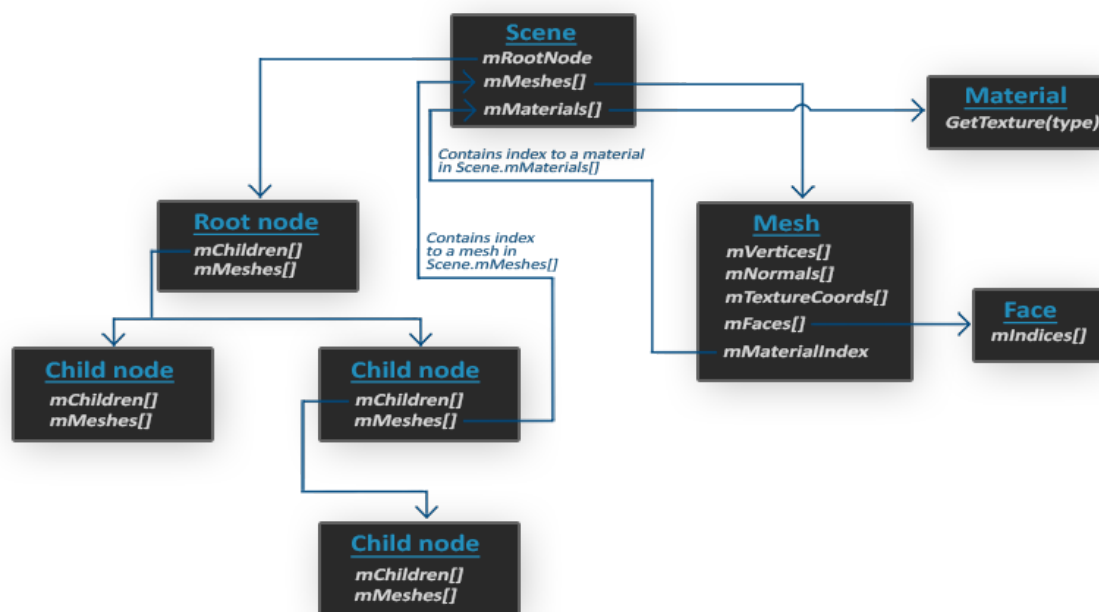
Assimp یک کتابخانه برای بارگذاری و پردازش صحنه‌های هندسی از فرمت‌های مختلف است. می توان با استفاده از آن مواردی همچون مش‌های استاتیک و یا اسکلتونی، مواد<sup>۴</sup>، انیمیشن های اسکلتونی و داده‌های بافت را از فایل بارگذاری کرد. زمانی که این مدل‌ها بارگذاری می شوند این کتابخانه آن‌ها را در ساختاری به شکل زیر ذخیره می کند و بعد از آن می توان از این ساختار، داده‌های مورد نظر خود را خواند و از آن‌ها استفاده کرد. [۱۶] [۱۷]

<sup>۱</sup>Contexts

<sup>۲</sup>Events

<sup>۳</sup>OpenGL Shading Language(GLSL)

<sup>۴</sup>Materials



شکل ۵-۱ - ساختار کلاس‌های کتابخانه‌ی Assimp [۱۷]

## ۶-۱-۵ 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. [۱۸]

## ۲-۵ پیاده‌سازی

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

## ۱-۲-۵ 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.

## ۲-۲-۵ 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<sup>۱</sup> 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<sup>۲</sup> صورت می‌گیرد. در نهایت گفتیم که آنچه برای کارت گرافیک اهمیت دارد دریافت مثلث‌ها است. بنابراین باید به طریقی بگوییم کدام رئوس با اتصال به یکدیگر مثلث تشکیل می‌دهند. اینکار نیز با استفاده از آرایه‌ی EBOs<sup>۳</sup> صورت می‌گیرد.

## ۳-۲-۵ اسکلت شخصیت

اسکلت یک شخصیت به صورت مجموعه‌ای از مفاصل که به صورت سلسله‌مراتبی به یکدیگر 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.

## ۴-۲-۵ 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 اگر 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 هر کدام از مفاصلی که به آن متصل است را بدست آوریم، سپس انتقال نهایی برابر مجموع وزن‌دار تمامی این انتقال‌ها خواهد بود.

<sup>۱</sup>Vertex Buffer Objects

<sup>۲</sup>Vertex Array Objects

<sup>۳</sup>Element Buffer Objects

## ۵-۲-۵ 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 داده شد همراه این اسکلت حرکت می‌کند.

انیمیشن‌ها از طریق کلاسی به اسم Animation Clip مدل‌سازی شده‌اند. این کلاس شامل آرایه‌ای از ژست‌های 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.

## ۵-۲-۶ 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، انیمیشن‌ها 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.

## ۵-۲-۷ الگوریتم پخش‌کننده‌ی انیمیشن

هر شخصیت درون بازی، اگر از نوع شخصیت اسکلتونی باشد، دارای یک پخش‌کننده‌ی 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 پخش کننده به روزرسانی می کنیم. سپس برای بدست آوردن زمان فعلی انیمیشن می توان از فرمول زیر استفاده کرد

$$\text{CurrentAnimationTime} = \text{CurrentTime} - \text{StartTimeForCurrentAnimation}$$

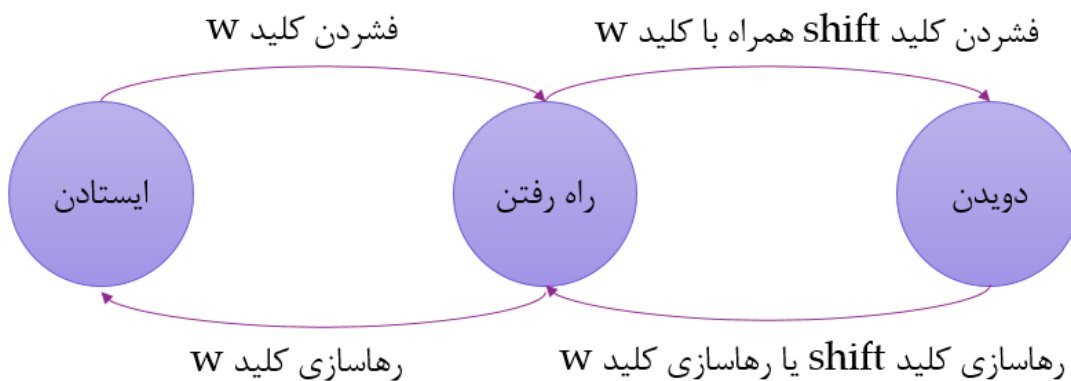
در نهایت با استفاده از این مقدار می توان ژست مورد نظر را از داخل کلیپ انیمیشنی بدست آورد. در نهایت نیز این ژست را بر روی اسکلت شخصیت اعمال می کنیم.

#### ۸-۲-۵ ماشین حالت انیمیشن

یکی از روش های ترکیب انیمیشن های 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 می کنند. زمانی که از ماشین حالت استفاده می شود سیستم می تواند در هر لحظه تنها در یکی از این حالات قرار گیرد. البته سیستم می تواند با دریافت ورودی از یک حالت به حالت دیگری رود.

دلیل استفاده از ماشین حالت متناهی برای سیستم انیمیشن این است که همانگونه که گفتیم، کلیپ های انیمیشنی، شامل ویدیوهای کوتاهی هستند که یک حالت مشخصی از شخصیت را بیان می کنند. در یک بازی، با توجه به ورودی بازیکن، شخصیت درون بازی می تواند در حالت های متفاوتی قرار گیرد. با استفاده از ماشین حالت می توان به تمامی این حالت ها رسیدگی کرد.

به عنوان مثال، تصویر زیر نشان دهنده ی یک ماشین حالت برای حرکت شخصیت است. شخصیت در ابتدا در حالت ایستاده قرار دارد و با گرفتن ورودی های مختلف از کیبورد، می تواند به حالت های دیگری رود.



شکل ۲-۵ - ماشین حالت برای حرکت شخصیت

برای پیاده سازی ماشین حالت متناهی، این سیستم به سه کلاس کلی شکسته شده است. کلاس AnimationStateMachine که وظیفه ی مدیریت حالت ها و انتقال از یک حالت به حالت دیگری را دارد. کلاس AnimationState که نشان دهنده ی



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.

## ۹-۲-۵ 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<sup>۱</sup>

۲. حالت در حال انتقال<sup>۲</sup>

۳. حالت اتمام انتقال<sup>۳</sup>

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<sup>۴</sup> را بازگردانند، ماشین به وضعیت دوم که وضعیت در حال 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 پخش‌کننده 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())
        {
            transitionStatus = TransitionStatus::transitioning;
            currentState = animationStatesMap.at(transition->to);
            TransitionFromPose = animator->GetPoseAtCurrentTime();
        }
    }
}
```

---

<sup>۱</sup>Normal

<sup>۲</sup>Transitioning

<sup>۳</sup>Finished

<sup>۴</sup>True

```

        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] “Animation wikipedia”, <https://en.wikipedia.org/wiki/Animation>.
- [2] Barczak, A. M. and Woźniak, H., “Comparative study on game engines”, *Studia Informatica. Systems and Information Technology. Systemy i Technologie Informacyjne*, No. 1-2, 2019.
- [3] Lasseter, J., “Principles of traditional animation applied to 3d computer animation”, *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987.
- [4] Gregory, J., *Game Engine Architecture*, A K Peters/CRC Press, 3rd ed. , 2018.
- [5] “Unreal engine wikipedia”, [https://en.wikipedia.org/wiki/Unreal\\_Engine](https://en.wikipedia.org/wiki/Unreal_Engine).
- [6] “Unreal engine blueprint”, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/>.
- [7] “Computer graphics wikipedia”, [https://en.wikipedia.org/wiki/Computer\\_graphics\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Computer_graphics_(computer_science)).
- [8] “Using opengl”, [https://www.khronos.org/opengl/wiki/Getting\\_Started](https://www.khronos.org/opengl/wiki/Getting_Started).
- [9] “Opengl state machine”, <https://learnopengl.com/Getting-started/OpenGL>.
- [10] “Phone shading learnopengl”, <https://learnopengl.com/Lighting/Basic-Lighting>.
- [11] “Phong shading wikipedia”, [https://en.wikipedia.org/wiki/Phong\\_reflection\\_model](https://en.wikipedia.org/wiki/Phong_reflection_model).
- [12] “Unreal engine architecture”, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/>.
- [13] “Unreal engine components”, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Components/>.
- [14] “Glfw”, <https://github.com/glfw/glfw>.
- [15] “Glad”, <https://github.com/Dav1dde/glad>.
- [16] “Assimp”, <https://assimp-docs.readthedocs.io/en/v5.1.0/about/introduction.html>.
- [17] “Assimp class hierarchy”, <https://learnopengl.com/Model-Loading/Assimp>.
- [18] “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.