

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.....	هشت
فهرست جدول ها.....	نه
فهرست الگوریتم ها.....	ده
چکیده.....	۱
فصل اول: مقدمه	۲
۱-۱ موتور بازی سازی.....	۲
۲-۱ موتور بازی سازی آنریل.....	۳
۳-۱ انیمیشن های کامپیوتری.....	۴
فصل دوم: انیمیشن های اسکلتونی	۵
۱-۲ مدل اسکلتونی.....	۵
۲-۲ شبکه ی ^۱ چندضلعی.....	۶
۱-۲-۲ راس.....	۶
۲-۲-۲ لبه.....	۶
۳-۲-۲ وجه.....	۶
۴-۲-۲ چندضلعی.....	۷
۳-۲ مدل.....	۷
۴-۲ زیرمش ^۲	۷
۵-۲ ماده ^۳	۷
۶-۲ بافت ^۴	۸
۷-۲ اسکلت.....	۸
۸-۲ Skinning.....	۸
۹-۲ ژست شخصیت.....	۸
۱۰-۲ کلیپ های انیمیشنی.....	۹
۱۱-۲ ترکیب انیمیشن.....	۱۰
فصل سوم: آشنایی با موتور بازی	۱۱
۱-۳ ساخت پروژه.....	۱۱
۲-۳ ویرایشگر آنریل.....	۱۴
۳-۳ زبان برنامه نویسی.....	۱۵
۴-۳ رایج ترین اصطلاحات.....	۱۵

¹ Mesh

² Sub-Mesh

³ Material

⁴ Texture

۱۵	۱-۴-۳ پروژه
۱۶	۲-۴-۳ شی
۱۶	۳-۴-۳ بازیگران ^۱
۱۶	۴-۴-۳ تغییر نوع داده ^۲
۱۷	فصل چهارم: معماری موتور بازی سازی آنریل
۱۷	۱-۴ Actors و UObject
۱۸	۲-۴ Components
۲۰	فصل پنجم: پیاده سازی
۲۰	۱-۵ ابزارها
۲۰	۱-۱-۵ OpenGL
۲۱	۲-۱-۵ GLFW
۲۱	۳-۱-۵ GLAD
۲۱	۴-۱-۵ GLM
۲۱	۵-۱-۵ Assimp
۲۲	۶-۱-۵ stb
۲۲	۲-۵ پیاده سازی
۲۲	۱-۲-۵ نمایش مدل گرافیکی
۲۳	۲-۲-۵ قرار گیری مدل سه بعدی در کارت گرافیک
۲۳	۳-۲-۵ اسکلت شخصیت
۲۳	۴-۲-۵ اتصال اسکلت و مدل سه بعدی
۲۴	۵-۲-۵ انیمیشن
۲۴	۶-۲-۵ پخش کننده ی انیمیشن
۲۴	۷-۲-۵ الگوریتم پخش کننده ی انیمیشن
۲۵	۸-۲-۵ ماشین حالت انیمیشن
۲۶	۹-۲-۵ به روز رسانی ماشین حالت انیمیشن
۲۸	پیوست ها
۲۹	مراجع

¹ Actors

² Casting

فهرست شکل ها

عنوان	صفحه
شکل ۳-۱: مرورگر پروژه های آنریل	۱۲
شکل ۳-۲: قالب پروژه های آنریل	۱۳
شکل ۳-۳: تنظیم پروژه های آنریل	۱۴
شکل ۳-۴: ویرایشگر آنریل	۱۴
شکل ۴-۱: تصویر UML کلاس های پایه ی موتور بازی سازی آنریل	۱۹
شکل ۵-۱: ساختار کلاس های کتابخانه ی Assimp [۱۲]	۲۲
شکل ۵-۲: ماشین حالت برای حرکت شخصیت	۲۵

فهرست جدول ها

صفحه

عنوان

جدول ۱-۳: مدل های تبدیل ۱۵

فهرست الگوریتم‌ها

چکیده

کلمات کلیدی

فصل اول

مقدمه

۱-۱ 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. [۱] 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. [۲] 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: [۱]

- 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، ویژگی‌های شخصیت‌های بازیکنان، هوش مصنوعی و اشیاء موجود در دنیای مجازی و همچنین وظایف و اهداف بازیکنان را تعریف می‌کند.
- ابزارهای نرم‌افزاری: وظیفه اصلی این ابزارها افزایش راندمان و سرعت تولید بازی با موتور بازی‌سازی است. آن‌ها توانایی اضافه کردن بسیاری از عناصر مختلف را به بازی‌ها، از انیمیشن و جلوه‌های صوتی گرفته تا الگوریتم‌های هوش مصنوعی، را فراهم می‌کنند.

۲-۱ موتور بازی‌سازی آنریل

اولین نسل موتور بازی‌سازی آنریل توسط تیم سوینی، بنیانگذار اپیک گیمز^۱، توسعه یافت. سوینی در سال ۱۹۹۵ شروع به نوشتن این موتور برای تولید بازی تیراندازی اول شخصی به اسم غیرواقعی^۲ کرد. نسخه‌ی دوم موتور بازی‌سازی آنریل در سال ۲۰۰۲ منتشر شد. نسخه سوم نیز در سال ۲۰۰۴ پس از اینکه حدود ۱۸ ماه در حال توسعه بود منتشر شد. در این نسخه، معماری پایه‌ای موجود در نسخه‌ی اول مانند طراحی شی گرا، اسکریپت‌نویسی مبتنی بر داده و رویکرد نسبتاً ماژولار نسبت به زیرسیستم‌ها وجود داشت. اما برخلاف نسخه دوم که از یک خط لوله با عملکرد ثابت^۳ استفاده می‌کرد، این نسخه به صورتی طراحی شده بود تا بتوان قسمت‌های سایه‌زنی سخت‌افزاری^۴ را برنامه‌نویسی کرد. موتور بازی‌سازی آنریل ۴ در سال ۲۰۱۴ در کنفرانس توسعه‌دهندگان بازی^۵ منتشر شد. این نسخه با طرح کسب و کار 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 ۵^۶ و ایکس‌باکس سری X/S^۷ 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. [۳]

^۱Epic Games

^۲Unreal

^۳fixed-function pipeline

^۴shader hardware

^۵GDC

^۶PlayStation 5

^۷Xbox Series X/S

۳-۱ انیمیشن‌های کامپیوتری

فصل دوم

انیمیشن‌های اسکلتونی

انیمیشن اسکلتونی تکنیکی در انیمیشن‌های 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 سلسله‌مراتبی از قطعات به هم پیوسته است که به هر قطعه یک مفصل گویند. در این فصل به بررسی این دوبخش و تکنیک‌های موجود در انیمیشن‌های اسکلتونی خواهیم پرداخت

۱-۲ مدل اسکلتونی

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

۲-۲ شبکه‌ی^۱ 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^۲ 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.

۱-۲-۲ 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.

۲-۲-۲ لبه

ارتباط بین دو راس را لبه گویند.

۳-۲-۲ 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.

¹Mesh

²renderer

۴-۲-۲ 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.

۳-۲ مدل

مدل^۱ 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.

۴-۲ زیرمش^۲

چندضلعی‌های دارای یک نوع ماده^۳ 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^۴ 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.

۵-۲ ماده^۵

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

^۱ گاهی به جای استفاده از واژه‌ی مدل، از واژه‌ی مش هم استفاده می‌شود.

^۲ Sub-Mesh

^۳ Material

^۴ Render

^۵ Material

۲. میزان براقی شی

۳. رنگ (بافت) شی

۴. سایه زنی پیکسلی یا راسی^۱

۶-۲ بافت^۲

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.

۷-۲ 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.

۸-۲ 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.

۹-۲ 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

¹Vertex or Pixel shader

²Texture

مدل در یک ژست به خصوص قرار دارد که به آن ژست حالت اتصال^۱ یا ژست مرجع^۲ 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^۳ 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^۴ 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^۵ گفته می شود.

۲-۱۰ 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. [۲]

¹Bind Pose

²Reference Pose

³T Pose

⁴Local Pose

⁵Global Pose

۱۱-۲ 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. از ترکیب می‌توان برای یافتن یا حالت میانی بین دو حالت شناخته شده در زمان‌های مختلف 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

آشنایی با موتور بازی

۱-۳ ساخت پروژه

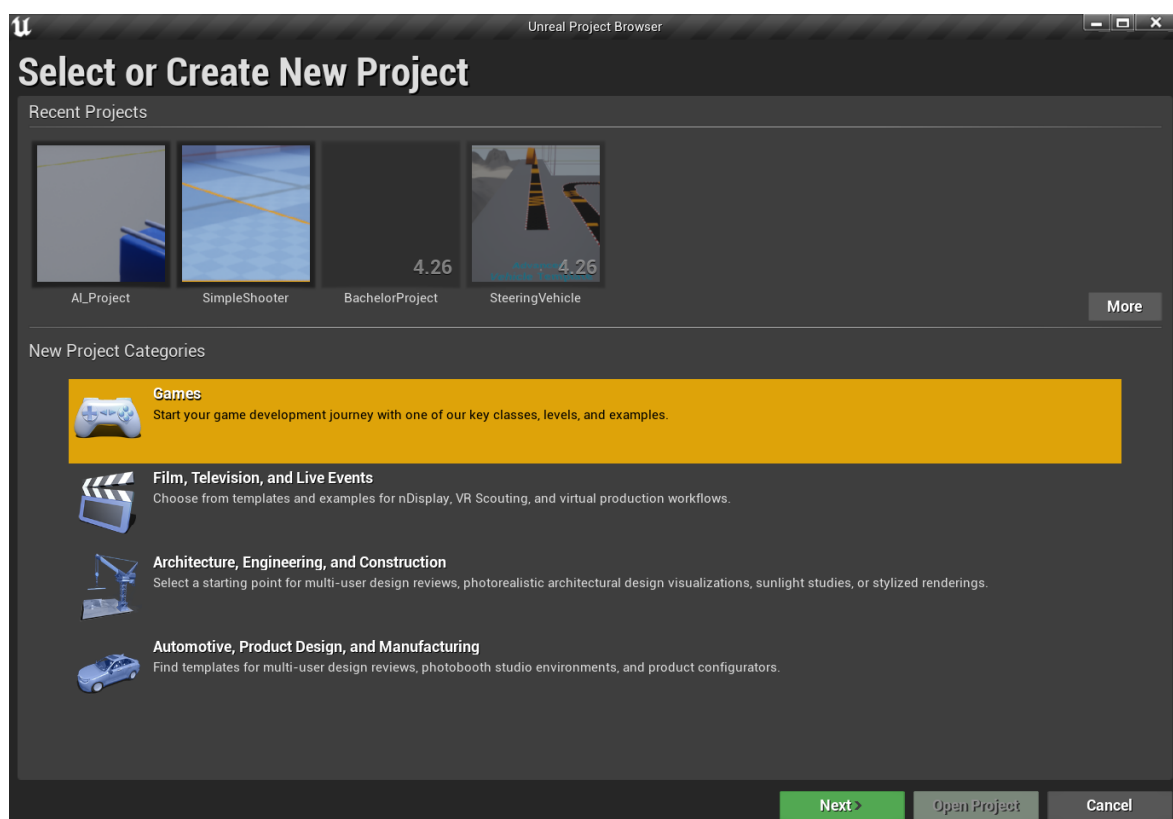
برای اینکه بتوان از موتور بازی سازی آنریل استفاده کرد ، ابتدا باید پروژه ای متناسب با کاری که می خواهیم انجام دهیم، بسازیم.

برای ساختن پروژه به صورت زیر عمل می کنیم.

ابتدا مرورگر آنریل انجین را باز می کنیم.

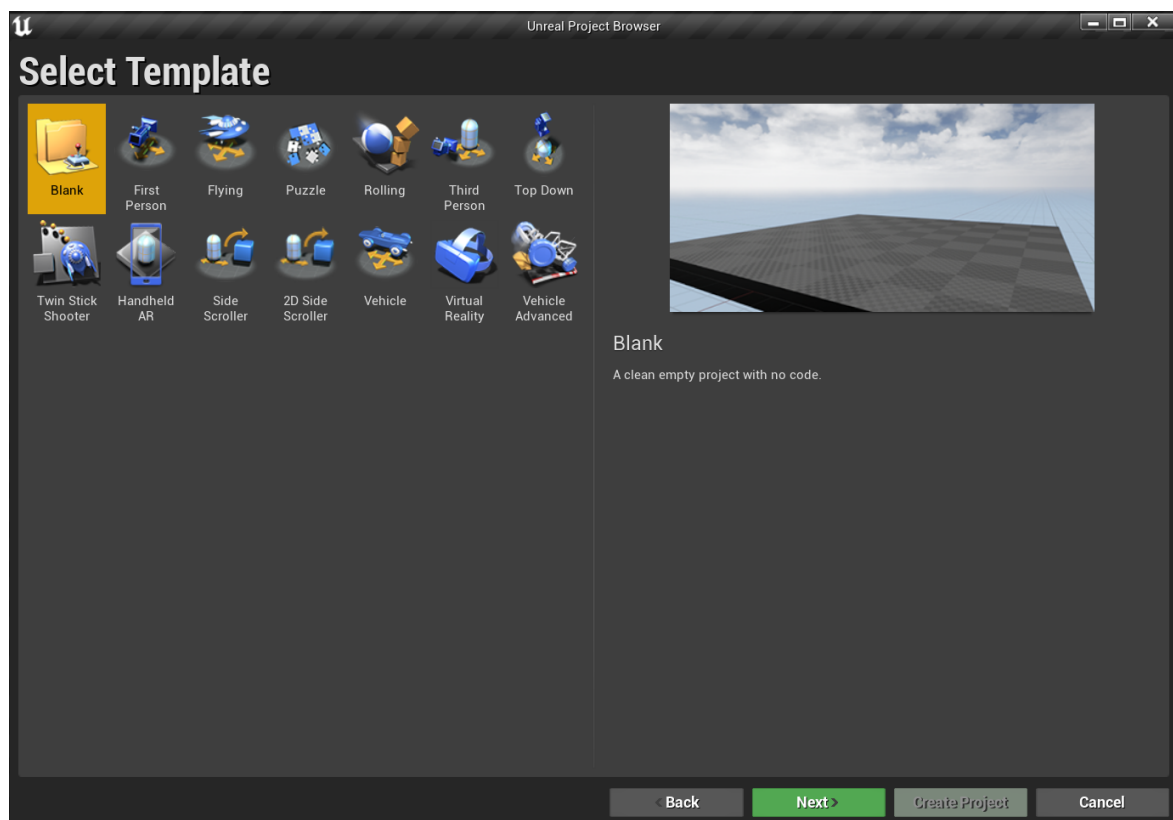
سپس با انتخاب دسته بندی مناسب و انتخاب گزینه ی بعدی به صفحه ی زیر هدایت می شویم. در اینجا زیر مجموعه ی

بازی را انتخاب شده است.



شکل ۳-۱ - مرورگر پروژه‌های آنریل

در اینجا قالب مناسب را انتخاب کرده و گزینه‌ی بعدی را کلیک می‌کنیم.



شکل ۲-۳- قالب پروژه‌های آنریل

در اینجا پس از انجام تنظیمات اولیه پروژه گزینه‌ی ایجاد پروژه را کلیک کرده و پروژه ساخته می‌شود.

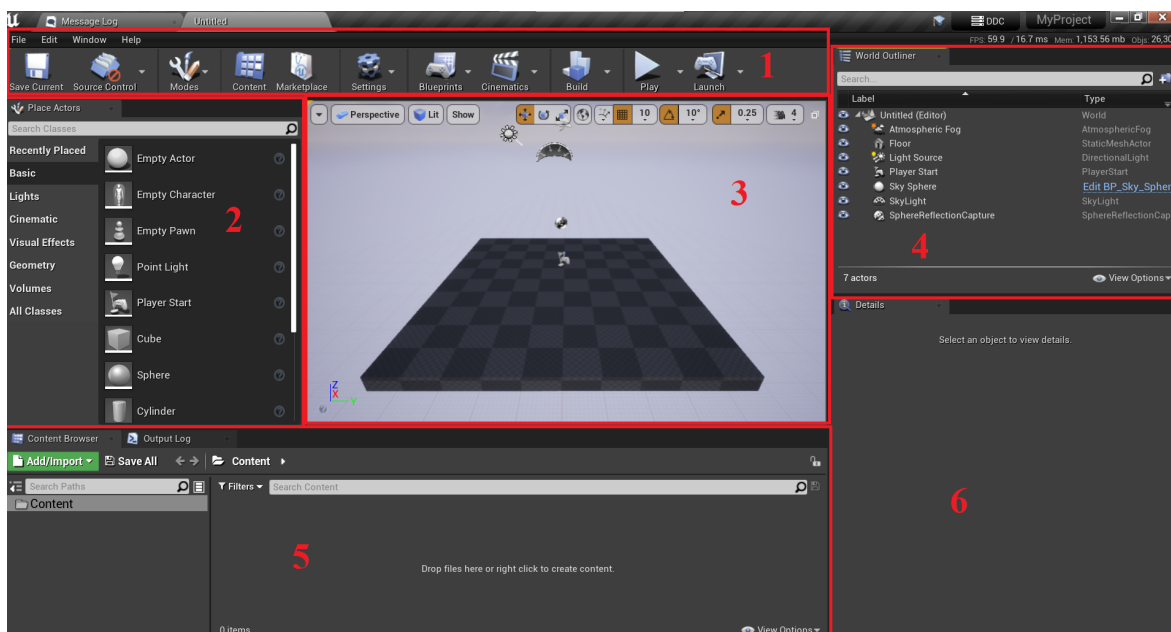


ردیابی اشعه

شکل ۳-۳ - تنظیم پروژه‌های آنریل

۲-۳ ویرایشگر آنریل

زمانی که پروژه ساخته می‌شود، ویرایشگر آنریل باز می‌شود. این ویرایشگر شامل پنل‌های مختلفی است که در عکس شماره زده شده و هر شماره نیز در جدول زیر توضیح داده شده است.



شکل ۳-۴ - ویرایشگر آنریل

جدول ۳-۱ - مدل‌های تبدیل.

شماره	نام	توضیح
۱	نوار ابزار	شامل توابع مختلفی است که به صورت معمول استفاده می‌شود
۲	بازیگران	می‌توان از این قسمت، بازیگر مناسب خود را انتخاب کرده و در صحنه‌ی بازی قرار داد
۳	درگاه دید ^۱	از طریق این پنل می‌توان اشیاء را در محیط قرار داد و بر روی اشیاء قرار گرفته شده کلیک کرد.
۴	طرح کلی جهان ^۲	تمامی اشیاء در مرحله فعلی را نشان می‌دهد. می‌توان اشیاء را با قرار دادن آن‌ها در پوشاها سازماندهی کرد. همچنین قابلیت جستجو بر اساس نوع را نیز دارد.
۵	مرورگر محتوا	این پنل تمامی فایل‌های پروژه را نمایش می‌دهد. همچنین می‌توان با ایجاد پوشه فایل‌ها را دسته بندی و مرتب کرد. علاوه بر این با استفاده از نوار جستجو می‌توان فایل موردنظر خود را پیدا کرد
۶	جزئیات	این پنل برای نمایش یا تغییر ویژگی‌های شیء انتخاب شده است. با تغییر ویژگی‌ها تنها ویژگی شیء انتخاب شده تغییر پیدا می‌کند.

۳-۳ زبان برنامه‌نویسی

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.

[۴]

۳-۴ رایج‌ترین اصطلاحات

در این بخش رایج‌ترین اصطلاحات مورد استفاده در هنگام کار با موتور بازی‌سازی آنریل را بررسی می‌کنیم.

۳-۴-۱ پروژه

یک پروژه آنریل، شامل تمامی محتوای بازی است. محتوا می‌تواند خود به چندین پوشه که بر روی دیسک قرار دارند، تقسیم شود. بدیهی است که می‌توان این پوشه‌ها را به صورت دلخواه نامگذاری و سازماندهی کرد. پنل مرورگر محتوا داخل ویرایشگر آنریل همان ساختار راهنمای موجود در پوشه Project که بر روی دیسک قرار دارد را نشان می‌دهد.

هر پروژه دارای یک پرونده‌ی uproject. متناظر به خود است. این فایل نحوه ایجاد، باز کردن یا ذخیره یک پروژه است. بنابراین می‌توان چندین پروژه مختلف ایجاد کرد و به صورت موازی بر روی آن‌ها کار کرد.

۳-۴-۲ شی

اشیاء پایه‌ای ترین کلاس موجود در آنریل هستند. آنها مانند بلوک‌های سازنده عمل می‌کنند و دارای بسیاری از عملکردها و توابع مورد نیاز برای دارایی‌ها Assets هستند.

۳-۴-۳ بازیگران^۱

هر شی‌ای را که بتوان بر روی صحنه قرار داد مانند دوربین، مش استاتیک، محل شروع بازی و ... را بازیگر^۲ می‌گویند. بازیگران از تبدیل‌های سه‌بعدی مانند انتقال، دوران و تغییر مقیاس پشتیبانی می‌کنند. آنها را می‌توان از طریق کدهای گیمپلی (C++ یا برنامه‌کار^۳) ایجاد کرد و یا از بین برد.

۳-۴-۴ تغییر نوع داده^۴

تغییر نوع داده، عملی است که طی آن بازیگری از یک کلاس را گرفته و سعی می‌کند به گونه‌ای رفتار کند که گویی از کلاس دیگری است. این عمل ممکن است موفقیت آمیز باشد و یا شکست بخورد. در صورت موفقیت آمیز بودن می‌توان به توابع کلاسی که به آن تغییر داده شده دستیابی پیدا کرد.

¹ Actors

² Actor

³ Blueprint

⁴ Casting

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^۱ نمونه هایی از کلاس هایی هستند که از UObject ها ارث می برند. بازیگران^۲ 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^۳ می گویند. بنابراین اجزاء یک نوع خاصی از اشیاء هستند که بازیگران می توانند آن ها را به عنوان یک زیرشیء^۴ 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

¹ Objects

² Actors

³ Component

⁴ 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. [۵، ۶]

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^۱ مش‌ها^۲، 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^۳: 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^۴: 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^۵: 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^۶ یا همپوشانی^۷ با اشیاء فیزیکی استفاده می‌شود. این کلاس می‌تواند شامل مش‌های استاتیک^۸ یا اسکلتی^۹، اسپرایت‌ها یا بیلبردها، سیستم‌های ذرات^{۱۰} و همچنین حجم برخورد^{۱۱} جعبه، کپسول و کره شود.

[۶]

^۱Render

^۲Meshes

^۳ActorComponent

^۴SceneComponent

^۵Primitive Components

^۶collide

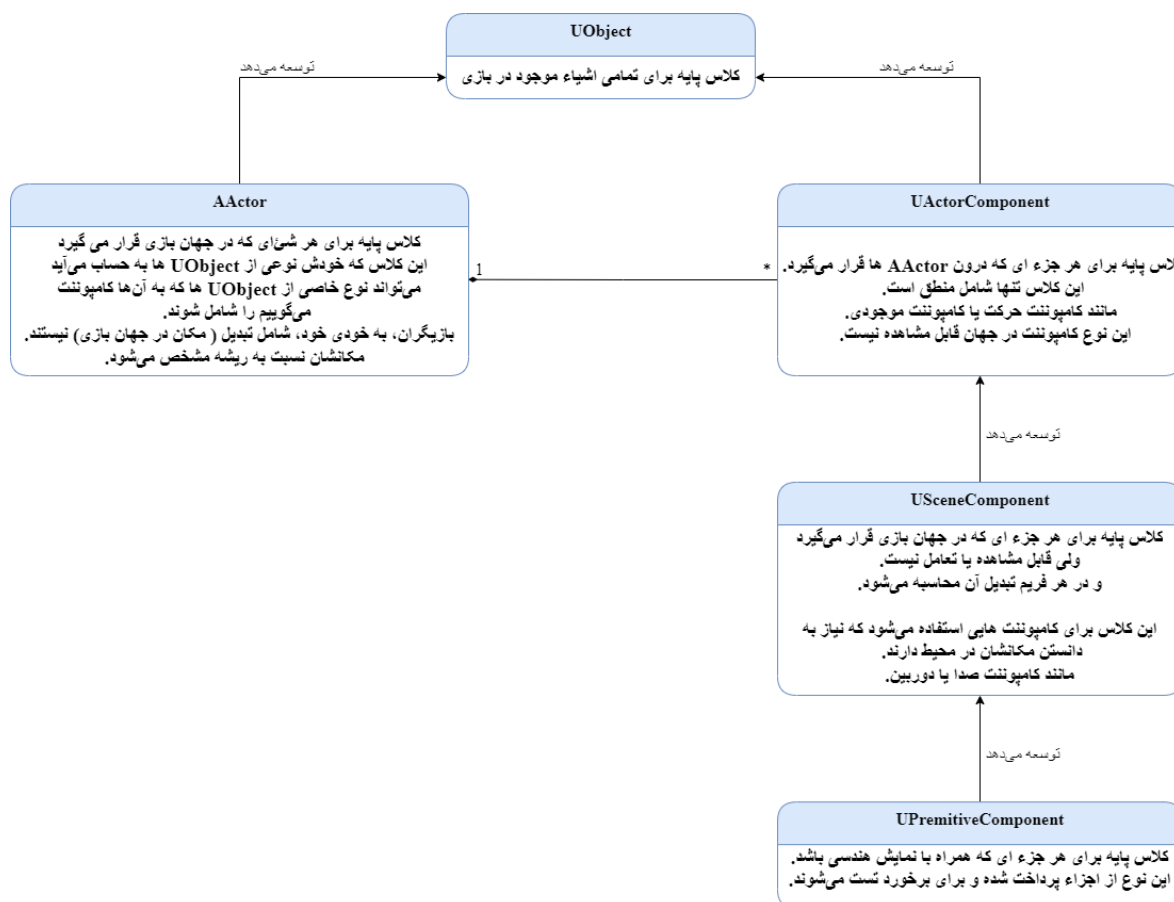
^۷overlap

^۸static mesh

^۹skeletal mesh

^{۱۰}particle systems

^{۱۱}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^۱ 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. [۷]

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

^۱API

context فعلی اعمال می شود و سپس رندر می شود. [۷] [۸]

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^۱ و سطوح، خواندن ورودی و مدیریت رویدادها^۲ را ارائه می کند. این کتابخانه از سیستم عامل‌های ویندوز، مک و لینوکس و سیستم‌های مشابه یونیکس پشتیبانی می کند. [۹]

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. [۱۰]

GLM ۴-۱-۵

GLM یک کتابخانه‌ی ریاضی برای نرم افزارهای 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 شده‌اند که بسیار به GLSL نزدیک باشند.

Assimp ۵-۱-۵

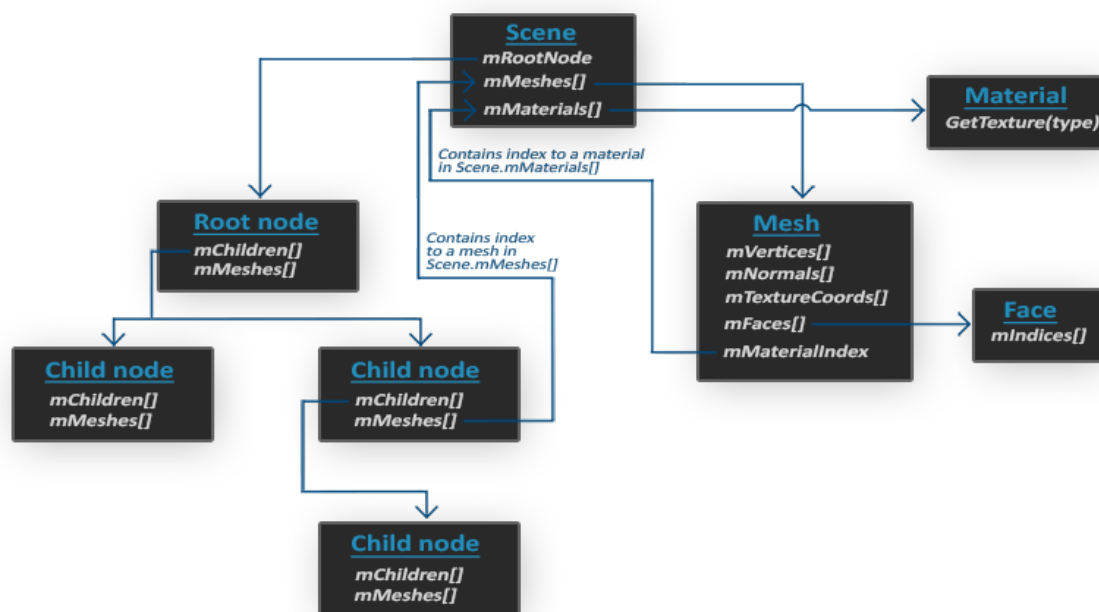
Assimp یک کتابخانه برای بارگذاری و پردازش صحنه‌های هندسی از فرمت‌های مختلف است. می توان با استفاده از آن مواردی همچون مش‌های استاتیک و یا اسکلتونی، مواد^۴، انیمیشن های اسکلتونی و داده‌های بافت را از فایل بارگذاری کرد. زمانی که این مدل‌ها بارگذاری می شوند این کتابخانه آن‌ها را در ساختاری به شکل زیر ذخیره می کند و بعد از آن می توان از این ساختار، داده‌های مورد نظر خود را خواند و از آن‌ها استفاده کرد. [۱۱] [۱۲]

^۱Contexts

^۲Events

^۳OpenGL Shading Language (GLSL)

^۴Materials



شکل ۵-۱ - ساختار کلاس‌های کتابخانه‌ی Assimp [۱۲]

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. [۱۳]

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 سه بعدی 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 سه بعدی ما را در خود جای می‌دهد. همانطور که در بخش ۲-۱ اشاره شد، رئوس علاوه بر اینکه شامل اطلاعات موقعیت مکانی در محیط سه بعدی 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 کدام رئوس با اتصال به یکدیگر مثلث تشکیل می‌دهند. اینکار نیز با استفاده از آرایه‌ی EBOs^۳ صورت می‌گیرد.

۳-۲-۵ اسکلت شخصیت

اسکلت یک شخصیت به صورت مجموعه‌ای از مفاصل که به صورت سلسله مراتبی 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 سه بعدی

اصطلاحی که برای اتصال اسکلت و مدل سه بعدی 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.

^۱Vertex Buffer Objectss

^۲Vertex Array Objects

^۳Element 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.

۶-۲-۵ 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.

۷-۲-۵ 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

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

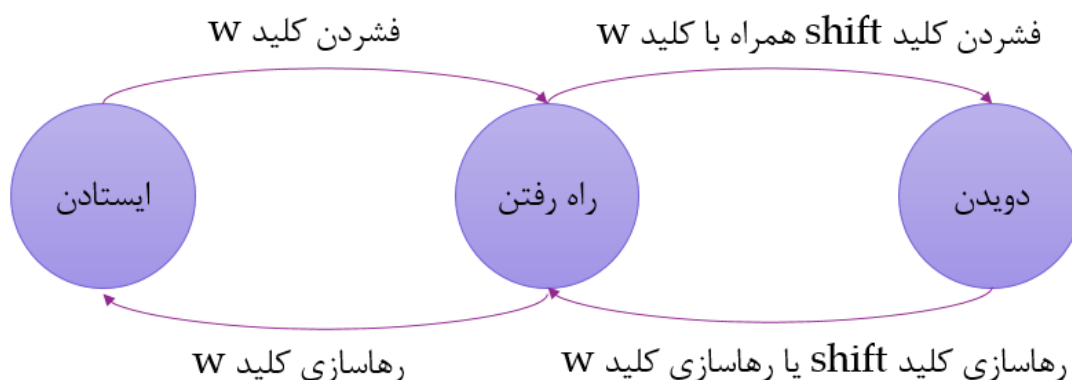
در نهایت با استفاده از این مقدار می توان ژست مورد نظر را از داخل کلیپ انیمیشنی بدست آورد. در نهایت نیز این ژست را بر روی اسکلت شخصیت اعمال می کنیم.

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^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 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();
            TransitionToPose = currentState->GetAnimClip()->GetPoseForCurrentFrame(0);
            currentTime = 0;
            transitionTime = transition->transitionTime;
        }
    }
}
```

¹Normal

²Transitioning

³Finished

⁴True

```
        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] 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.
- [2] Gregory, J., *Game Engine Architecture*, A K Peters/CRC Press, 3rd ed. , 2018.
- [3] “Unreal engine wikipedia”, https://en.wikipedia.org/wiki/Unreal_Engine.
- [4] “Unreal engine blueprint”, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/>.
- [5] “Unreal engine architecture”, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/>.
- [6] “Unreal engine components”, <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/Actors/Components/>.
- [7] “Using opengl”, https://www.khronos.org/opengl/wiki/Getting_Started.
- [8] “Opengl state machine”, <https://learnopengl.com/Getting-started/OpenGL>.
- [9] “Glfw”, <https://github.com/glfw/glfw>.
- [10] “Glad”, <https://github.com/Dav1dde/glad>.
- [11] “Assimp”, <https://assimp-docs.readthedocs.io/en/v5.1.0/about/introduction.html>.
- [12] “Assimp class hierarchy”, <https://learnopengl.com/Model-Loading/Assimp>.
- [13] “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.