ششمین کنفرانس بین‌المللی
«بازی‌های رایانه‌ای؛ فرصت‌ها و چالش‌ها»
۳۰ بهمن و ۱ اسفند ۱۳۹۹ – دانشگاه اصفهان

بهینه سازی در توسعه بازی برای بهبود عملکرد منابع سخت افزاری

محمد مهدی کرمانی[1]*، محمد حسین رحیمی پور[2]
دانش آموخته کارشناسی مهندسی فناوری اطلاعات، دانشگاه علمی و کاربردی جامعه اسلامی کارگران، اصفهان، ایران، Email: (kermani_mohammad@hotmail.com)
دانش آموخته کارشناسی مهندسی کامپیوتر دانشگاه پیام نور ، اصفهان، ایران،
Email: (mohammadh1375128@gmail.com)

چکیده

عملکرد یک بازی می تواند باعث موفقیت و یا شکست در تجربه ی کاربری آن شود. اجرای یک بازی که به درستی بهینه سازی شده و احساس نرمی و واکنش پذیری مناسبی ایجاد می کند و در سیستم عامل های سطح پایین با منابع سخت افزاری ضعیف تر نیز اجرا می شود، مخاطبان بیشتری را به خود جذب می کند. به حداکثر رساندن عملکرد هر بازی می تواند با بازی دیگر متفاوت بوده و امری کاملاً نسبی باشد. به عبارت دیگر، نکات مربوط به بهینه سازی عملکرد ممکن است برای یک بازی مفید باشد و برای بازی دیگر کاملاً بی معنی و به روشی متفاوت برای بهینه سازی احتیاج باشد. تولید بازی در دنیای امروز بوسیله ی موتور های مختلفی انجام می شود که با توجه به آمار بالای توسعه دهندگانی که از موتور یونیتی استفاده می کنند محیط آزمایش و نتایج ما در این موتور بررسی شده است. اگرچه بسیاری از موضوعات مطرح شده بصورت مفهومی بوده و مستقل از موتور مورد نظر، می توان در تولید بازی بکار گرفت. برای اجرای بازی برروی هر دستگاهی نیازمند سخت افزار و سیستم عامل مناسب برای نمایش و تعامل روان هستیم. مهمترین چالش در توسعه و ارائه ی روان بازی بهینه سازی فرایندهای پشت صحنه است که تعیین کننده میزان سخت افزار مناسب است. هرچه بهینه سازی بهتر و کامل تر انجام شده باشد منابع سخت افزاری کمتری مانند پردازشگر، حافظه ی اصلی، پردازشگر و حافظه ی گرافیکی نیاز خواهد بود. همچنین در اندازه ی فایل و بسته ی خروجی نهایی و اشغال فضای ذخیره سازی نیز تاثیر بسزایی خواهد گذاشت. بهینه سازی باید از مراحل ابتدایی تولید محتویات گرافیکی و برنامه نویسی رعایت شود اما پس از اینکه محتویات به موتور بازی سازی وارد و مراحل چیدمان آغاز شد نیز باید به روش هایی ایرادات بهینه سازی را بررسی و مرتفع ساخت. این مقاله نگاهی تخصصی به تحلیل و بکارگیری روش ها و ابزارهای بهینه سازی بازی جهت ارتقاء عملکرد و روان بودن بازی بر روی دستگاه های هدف دارد که با مطالعه ی آن می توان تاثیرات شگفت انگیزی بر روی خروجی نهایی مطلوب ایجاد کرد.

کلمات کلیدی: موتور یونیتی، بهینه سازی عملکرد بازی، توسعه بازی های ویدیویی، افزایش نرخ فریم، بهینه سازی مصرف منابع سخت افزاری

۱– مقدمه

یکی از مهم ترین چالش های توسعه دهندگان بازی های ویدیویی بهینه سازی در عملکرد بازی جهت نمایش و تعامل روان است به این منظور دو مرحله ی کلی برای فرایند بهینه سازی در نظر گرفته شده است. مرحله ی اول تولید محتویات بازی است که عموما در نرم افزارهایی بجز موتور بازی تولید می شوند مانند محتویات سه بعدی، دو بعدی، صدا، انیمیشن، بافت ها، تصاویر و ویدیوها. مرحله ی دوم زمانی است که محتویات تولید شده وارد موتور بازی سازی شده اند و می خواهیم مراحل و

صحنه ها را چیدمان، طراحی و تنظیم کنیم. همچنین در این مرحله کدهای نوشته شده به بازی اضافه می شوند که بهینه بودن کدها نیز در قسمت های بعدی بررسی و راه کارهای مناسب ارائه می گردد. استاندارد های در نظر گرفته شده برای تولید محتویات دو بعدی، سه بعدی، صدا و انیمیشن مستقل از نرم افزاری هایی است که بوسیله ی آنها تولید می کنیم و می توانید با هر روش و هر ابزاری که به آن مسلّط هستید انجام دهید.

۲– مرحله اول (تولید محتوا)

هر یک از انواع محتویات شامل مدل های سه بعدی، دوبعدی، صدا، تصاویر و ویدیوها باید در مرحله ی تولید استاندارد و بهینه تولید شوند که در این میان محتویات گرافیکی از همه مهمتر هستند. در این بخش به جزئیات تولید هر یک از محتویات می پردازیم.

۲–۱– ساخت مدل های سه بعدی بهینه: کیفیت و جزئیات مدل های سه بعدی وابسته به پلت فرم هدف شما و قدرت سخت افزاری دستگاه های مورد نظر می تواند متفاوت باشد. منظور از کیفیت و جزئیات، تعداد راس ها و سه ضلعی های تشکیل دهنده ی سطوح اشکال سه بعدی است که هر چه تعداد آنها بیشتر باشد قطعا بار بیشتری برای سخت افزار ایجاد می کند. اینکه سطوح احجام سه بعدی از سه ضلعی، چهار ضلعی و یا چند ضلعی های بیشتر از آن ساخته شده باشد در پردازش آن تفاوتی ندارد و همه ی آنها بصورت سه ضلعی پردازش می شود. تنها توپولوژی آنها جهت تولید انیمیشن و ساخت بافت اهمیت دارد. بطور کلی استاندارد کاملاً دقیق و قطعی برای تعیین تعداد راس ها و سه ضلعی ها وجود ندارد و با گذر زمان هر چه سخت افزارها قوی تر می شوند امکان استفاده از احجام سه بعدی با جزئیات تر، بیشتر می شود ولی آمار گیری از نمونه های ساخته شده در بازی های مختلف نشان می دهد در کاراکتر شخص اول بازی که مهم ترین مدل سه بعدی محسوب می شود و بیشترین تعداد سه ضلعی را دارد حدودا بین ۱۵۰۰۰ الی ۲۵۰۰۰ سه ضلعی استفاده شده است و در حالتی که کاراکتر دارای لباس، وسایل، اسلحه و سایر متعلقات باشد این عدد به ۵۰۰۰۰ الی ۶۰۰۰۰ سه ضلعی افزایش می یابد. این اعداد در ساخت کارکتر ها NPC که نسبت به کاراکتر اصلی از اهمیت پایین تری برخوردار است به ۱۰۰۰۰ الی ۳۰۰۰۰ کاهش می یابد. اگرچه این اعداد مربوط به بازی های کامپیوتری و کنسولی مانند Xbox و Play Station است و برای نسخه های موبایلی بسیار متفاوت است. در نسخه های موبایلی ممکن است برای کاراکتر اصلی حداکثر به ۱۰۰۰۰ سه ضلعی برسد و برای NPC ها تا ۲۰۰ سه ضلعی نیز کاهش پیدا کند. استفاده از تکنیک Projection Mapping می تواند در نمایش با کیفیت مدل های کم حجم بسیار موثر باشد. البته کمی کار تولید را زمان بر و پرهزینه تر می کند زیرا که مدل ها دوبار ساخته می شوند. یک بار با کیفیت و جزئیات بالا و یک بار با جزئیات محدود تر، سپس بوسیله Projection Mapping جزئیات مدل با کیفیت در قالب بافت های مختلف به مدل کم کیفیت تر انتقال می یابد. از دیگر تکنیک های کنترل رئوس در رندر نهایی می توان به روش Level of Detail اشاره کرد که در آن از هر مدل دو یا چند نسخه ی کم کیفیت تر ایجاد می شود و با امکانات موجود در موتور بر مبنای فاصله ی نمایش آن، نسخه های مختلفی ارائه داده می شود تا در صورت امکان در فواصل بیشتر که جزئیات احجام کمتر دیده می شوند نسخه های کم حجم تر رندر گرفته شوند و مصرف منابع سخت افزاری کاهش پیدا کند.

۲–۲– بهینه سازی بافت ها: قطعا در ساخت بافت سایز تصویر بسیار پر اهمیت است یعنی اینکه در هر تصویر تعداد پیکسل ها در عرض و ارتفاع چه تعدادی باشند بر روی بزرگی فایل تاثیر زیادی دارد ولی نکاتی مانند نرخ بیت که ۸ بیت، ۱۶ بیت یا ۳۲ بیت باشد و یا فرمت فایل نیز پر اهمیت هستند. فرمت هایی که از کانال های رنگی بیشتری برخوردار هستند و یا دارای لایه های شفاف هستند سایز بیشتری خواهند داشت و از منابع سخت افزاری بیشتری استفاده می کنند. از طرف دیگر تصاویر

با اندازه های(Powered Two) یاهمان ۲ به توان n که عرض و ارتفاع آنها با یکدیگر برابرند بهتر از تصاویری هستند که از عرض و ارتفاع ۲ به توان n تشکیل شده اند ولی از عرض و ارتفاع برابر برخوردار نیستند و اگر اندازه ی تصویر بافت از هیچ یک از این قوانین پیروی نکرده باشد بدترین حالت ممکن است چرا که در این صورت موتور بازی کمترین میزان فشرده سازی را می تواند بر روی آنها انجام دهد. پس تلاش کنید UV ها و Mapping خود را با دقت در هر تصویر چینش کنید و تا جای ممکن از تمام پیکسل ها استفاده کنید. ساخت اطلس بافت از احجام مختلف بسیار در کاهش مصرف منابع سخت افزاری موثر است به این دلیل که چندین بافت از احجام مختلف در یک بافت مشترک چینش و آدرس دهی می شوند.

۲–۳– بهینه سازی انیمیشن ها: موتور یونیتی به صورت پیش فرض فقط تغییرات انیمیشنی حرکت، چرخش و تغییر اندازه در اشکال را درک می کند و از ویرایشگر های Skin و Morph نیز پشتیبانی می کند. مهم ترین اصول در ساخت انیمیشن بهینه، کاهش تعداد فریم های کلیدی و همچنین نرخ فریم است. اگر از ساخت انیمیشن در موتور استفاده می کنید دقت کنید که تنظیمات پیش فرض نرخ فریم یونیتی ۶۰ فریم بر ثانیه است که می توانید در صورت عدم لزوم این نرخ را به ۳۰ یا کمتر کاهش دهید. در احجام دارای استخوان هر چه تعداد استخوان ها کمتر باشد بار کمتری بر منابع سخت افزاری ایجاد می شود و هرچه تاثیر پذیری رئوس احجام از تعداد استخوان کمتری باشد می توان در موتور با تنظیم Skin Weights به حالت 1 Bone ویا 2 Bones بار اضافه بر منابع سخت افزاری را کاهش داد. سعی کنید از انیمیشن های تکرار پذیر (loop able) و یا با قابلیت Pong Ping بجای انیمیشن ها طولانی استفاده کنید. استفاده از روش ترکیب کلیپ انیمیشن در موتور اجازه می دهد که کلیپ های کمتری ساخته شود. بطور مثال بجای داشتن ۵ کلیپ انیمیشنِ راه رفتن، دویدن، تیر اندازی، راه رفتن و تیر اندازی، دویدن و تیر اندازی می توان از ۳ کلیپ انیمیشنِ راه رفتن، دویدن، تیر اندازی استفاده کرد و با بکارگیری امکان ترکیب، قسمت بالا تنه ی کلیپ انیمیشن تیر اندازی را با قسمت پایین تنه ی کلیپ انیمیشن راه رفتن و یا دویدن ترکیب کرد و دو کلیپ انیمیشن دیگر، بدون ایجاد حجم اضافه تر در خروجی نهایی تولید کرد.

۲–۴– بهینه سازی صدا: در ساخت فایل های صدا، در نظر گرفتن Sample Rate، Bit depth و تعیین کانال ها جهت بهترین کیفیت و کمترین حجم بسیار پر اهمیت است. استفاده از کلیپ های صوتی کوتاه با Sample Rate حداکثر ۴۸۰۰۰ و Bit Depth حداکثر ۳۲ می تواند تمام نیاز شما نسبت به کیفیت ایده ال باشد و در صورتی که محتویات کلیپ صوتی شما در کانال های چپ و راست متفاوت نیست بجای حالت stereo از حالت mono استفاده کنید. در صورتی که در تیم شما مهندس صدا و هنرمند آهنگ ساز وجود دارد و می تواند از امکانات audio mixer و استدیو صدا موتور بخوبی استفاده کند می توانید موسیقی های متنوع را با کمترین هزینه برای منابع سخت افزاری ایجاد کرده و به راحتی در زمان اجری بازی آنها را مدیریت کنید. همچنین موتور یونیتی این امکان را می دهد که با تبدیل صداهای استریو به مونو و تنظیم میزان فشرده سازی با حفظ کیفیت، حجم فایل های صوتی خود را بطور چشمگیری کاهش دهید. در موتور امکان بارگذاری فایل های صوتی در حافظه اصلی و یا پخش آنها به صورت جریانی (stream) وجود دارد که هر دو روش مزایا و معایبی دارند. از جمله مزایای پخش صدا بصورت جریانی، حذف زمان بارگذاری فایل صدا در زمان بارگذاری مرحله ی بازی بوده که به میزان قابل توجهی زمان بارگذاری را کاهش می دهد.

۳– مرحله دوم (پیاده سازی در موتور)

۳–۱– بهینه سازی در مواد و شیدرها: در واردات محتویات سه بعدی می توان تنظیمات مربوط به Material را به حالت standard(Legacy) گذاشت و گزینه ی بکارگیری مواد بیرونی (use external materials) را استفاده کرد سپس ساختار نام

دهی به متریال را به حالت By base texture name تنظیم کرد. مهم ترین مزیت این تنظیم این است که یونیتی در ازای هر شیء سه بعدی و متریال های آن که بر اساس نام texture های اعمال شده ساخت می شوند، ابتدا کل پوشه ی مربوط به متریال را جستجو می کند و در صورت وجود متریال هم نام، متریال جدید تولید نکرده و از همان متریال قبلی استفاده می کند. این روش در کاهش تعداد متریال ها و هزینه های سخت افزاری بسیار مفید است وکلیه ی احجامی که از یک نوع بافت استفاده می کند را براحتی در یک دسته بندی قرار داده و به GPU ارسال می کند. انتخاب نوع شیدر برای هر متریال بسیار مهم است زیرا هر چه شیدر، لایه های بیشتری از بافت ها و UV ها را پشتیبانی کند بار بیشتری ایجاد می کند. در یونیتی شیدرهای متنوعی بصورت پیش فرض گذاشته شده است که از ساده ترین حالت مانند Mobile/Defuse تا حالات پیشرفته ای مانند Autodesk Standard را پوشش می دهند. اگرچه امکان برنامه نویسی شیدر در یونیتی به راحتی امکان پذیر است ولی توصیه می شود تا جای ممکن از شیدر های پیچیده با توابع ریاضیاتی مانند sin، cos، pow استفاده نشود.

۳-۲- بهینه سازی نور ها و لایت مپ: نورهای realtime با قابلیت تولید سایه از نورهای همین نوع ولی بدون قابلیت تولید سایه بار بیشتری برای منابع سخت افزاری ایجاد می کنند. در این میان تولید Lightmap جهت ایجاد روشنای و سایه های ثابت بهترین گزینه برای اجسام بدون تغییر در صحنه هستند اما استانداردها و پارامترهای مهمی در تهیه ی یک Lightmap چشم نواز و طبیعی، با کیفیت مناسب و حجم کم وجود دارد. از جمله texel ها که هر چه تعداد آن بیشتر باشد می تواند کیفیت مطلوب تری از نور و سایه های ثابت ایجاد کند ولی تعداد و سایز map ها را افزایش می دهد ویا indirect intensity که می تواند نقاطی که در نور مستقیم نیستند را روشن تر و یا تیره تر سازد ولی با افزایش آن زمان Bake کردن طولانی تر می شود. برای رسیدن به حالت مطلوب انجام مراحل زیر مفید و کارآمد است:

- گزینه ی Generate Lighmap UVs کلیه احجامی که در صحنه ثابت هستند را در قسمت واردات و در قسمت Model فعال کنید.
- پس از اضافه کردن مدل به صحنه، در قسمت inspector گزینه ی static آن را فعال کنید.
- در کامپوننت Mesh Renderer و زیر بخش Lightmapping می توانید گزینه ی Scale in Lightmap را بر اساس اهمیت نمایش از عدد ۱ به عددی بالاتر و یا پایین تغییر دهید. این موضوع کمک می کند نقاط کم اهمیت سایز کوچکتری در Lightmap ها داشته باشند (بعضا این عدد می تواند به صفر برسد).
- Lightmap Resolution را از اعداد کمتر از ۱۰ شروع کنید و در صورت لزوم به اعداد بالاتر تغییر دهید.
- تا جای ممکن از افکت مه (fog) استفاده نکنید.

۳-۳- بهینه سازی جلوه های بصری: به وسیله امکانات موجود در موتور نظیر Bloom، Ambient Occlusion، Color Adjustments، Auto Exposure، Depth of Field، Grain، Lens Distortion، Motion Blur، Vignette و چندین امکان دیگر ایجاد جلوه در صحنه از زیر مجموعه ی Post-Processing می توان جلوه های بسیار زیبا و واقع گرایانه ایجاد کرد البته همانطور که از نام Post-Processing پیدا است پردازش این جلوه ها در هنگام اجرا انجام می شود در نتیجه برای منابع سخت افزاری سربار ایجاد می کند و هرچه کیفیت و جزئیات بیشتری داشته باشند این سربار بیشتر خواهد بود. در سیستم های ذره ای تعداد ذرات در هر ثانیه، شکل فیزیکی ذارت، فیزیک ذارت، متریال و شیدر ذارت، طول عمر، انیمیشن و رفتار ذرات مهمترین عناصر تاثیر گذار بر روی منابع سخت افزاری است. هر چه تعداد ذرات موجود در صحنه در هر لحظه از زمان بیشتر باشد و یا

قابلیت های فیزیکی بیشتری داشته باشند و یا از شیدرهای پیچیده یا حجیم استفاده کرده باشند قطعا برای منابع سخت افزاری پر مصرف محسوب می شوند.

۳–۴– بهینه سازی فیزیک: اجسام با کامپوننت Collider نسبت به اجسام بدون این کامپوننت دارای بار بیشتری برای منابع سخت افزاری هستند. هرچه collider یک جسم از حالت Box به حالت های پیچیده تری مانند Mesh collider برود به لحاظ پردازشی سنگین تر خواهد بود. پس اگر مدل سه بعدی شما بصورت پیش فرض این کامپوننت را دارد ولی به آن نیاز ندارد حتما آن را از روی شیء حذف کنید. قطعا در بازی برای اینکه اجسام و کاراکتر ها از دیوار ها، سقف و کف عبور نکنند نیازمند اضافه کردن این کامپوننت به محیط هستیم ولی محیط طراحی شده قالبا دارای جزئیات زیادی است که الزاما در محاسبه ی سطوح قابل برخورد کارایی ندارند. به این منظور می توان یک یا چند مدل مجزا و بسیار کم حجم برای مشخص کردن محدوده ی قابل برخورد تولید و استفاده کرد سپس کامپوننت Mesh renderer آنها را حذف کنید. به طور اختصار هر چه اشکال فیزیکی ما از تعداد زوایای بیشتری برخوردار باشند و هر چه تعداد احجامی که در صحنه دارای فیزیک و تعاملات فیزیکی هستند بیشتر باشد، محاسبه ی آنها برای منابع سخت افزاری پر هزینه تر است.

۳–۵– دسته بندی ایستا و پویا: برای ترسیم یک شیء روی صفحه نمایش، موتور باید یک درخواست ترسیم به API هایی مانند OpenGL و یا DirectX صادر کند. درخواستهای ترسیم اغلب به لحاظ مصرف منابع سخت افزاری، پر مصرف محسوب می شوند به این معنی که در هر درخواست ترسیم API گرافیکی میزان قابل توجهی از منابع سخت افزاری را به کار گرفته و هزینه ی زیادی را برای عملکرد CPU ایجاد می کند. این موضوع در تغییر وضعیت اشیاء شدت می یابد مانند تغییر در ماده یک شیء که باعث اعتبار سنجی مجدد منابع و تکرار مراحل ترجمه در درایور گرافیکی می شود. در موتور یونیتی دو روش برای حل این مسئله وجود دارد. در دسته بندی پویا، راس های اشیاء و مش های کوچک را به CPU انتقال داده، راس های مشابه را با هم گروه کرده و سپس همه ی آنها را یکجا ترسیم می کند ولی در دسته بندی ایستا، اشیاء ثابت در صحنه را با یکدیگر ترکیب و به اشیاء بزرگتر تبدیل می کند و آنها را به روشی سریع تر رندر می گیرد. عمل دسته بندی خودکار در موتور مزایا و ایراداتی نسبت به دسته بندی دستی دارد. از جمله مزایای دسته بندی خودکار در مقایسه با دسته بندی دستی این است که در دسته بندی خودکار که توسط موتور انجام می شود اشیاء همچنان قادر خواهند بود که جداگانه ترسیم شوند و از جمله ایرادات دسته بندی خودکار توسط موتور، ایجاد سربار ذخیره سازی در حافظه در حالت دسته بندی ایستا بوده و همچنین ایجاد سربار در CPU در حالت دسته بندی پویا است. تنها اشیاء با مواد اشتراک گذاری شده را می توان با یکدیگر دسته بندی کرد از اینرو اگر می خواهید دسته بندی خوبی داشته باشید باید در ایجاد و اشتراک گذاری مواد در اشیاء به درستی هدف گذاری و تصمیمی گیری کنید. اگر دو ماده یکسان دارید که فقط در بافت متفاوت هستند، می توانید آن بافت ها را در یک بافت بزرگ ترکیب کنید. این فرایند را معمولاً اطلس گذاری بافت می نامند. وقتی بافت ها در یک اطلس قرار گرفتند، می توانید از یک ماده واحد استفاده کنید. اگر نیاز است که به خصوصیات یک ماده ی به اشتراک گذاشته شده از طریق اسکریپت دسترسی داشته باشید، لازم است که بجای استفاده از Renderer.material از Renderer.sharedMaterial استفاده کنید. اشیاء با قابلیت ایجاد سایه در هنگام رندر می توانند به هم متصل شوند حتی اگر مواد آنها متفاوت باشند. اشیاء با قابلیت ایجاد سایه در یونیتی حتی با مواد متفاوت می توانند از دسته بندی پویا استفاده کنند به شرطی که مقادیر موجود در خصوصیات عبور نور و ایجاد

سایه آنها یکسان باشد. برای مثال در جعبه های مشابه برای ایجاد سایه، داشتن بافت های متفاوت مهم نبوده و می توانند با یکدیگر دسته بندی شوند.

۳–۵–۱– دسته بندی پویا (اشیاء سه بعدی): موتور یونیتی می تواند اشیاء متحرک را درصورتی که مواد مشابه و سایر معیارهای لازم را داشته باشند به طور خودکار دسته بندی و ترسیم کند. این کار بصورت خودکار انجام شده و نیازی به تلاش اضافه نیست. دسته بندی پویا در ازای هر رأس از شیء ایجاد سربار خاصی می کند البته دسته بندی پویا تنها برروی اشیاء با حداقل ۳۰۰ رأس و حداکثر ۹۰۰ رأس اعمال می کند. اگر Shader مورد نظر شما از خصوصیات Vertex Position و Normal و تنها یک UV استفاده می کند می توانید تا ۳۰۰ رأس داشته باشید و اگر از Shader شما از UV0 و UV1 و همچنین از Vertex Position و Normal و Tangent استفاده می کند پس فقط می تواند تا ۱۸۰ رأس داشته باشید. اگر اشیاء یکسان دارای تغییرات در اندازه و یا قرینه سازی باشند در یک دسته بندی قرار نمی گیرند. استفاده از نمونه مواد مختلف در اشیاء یکسان سبب می شود که آنها در یک دسته بندی قرار نگیرند (به استثنای سایه ها که در یک دسته بندی قرار می گیرد). اشیاء با Lightmap، دارای پارامترهای بیشتری در رندر هستند. مانند شماره ی فهرست، اندازه و مکان هر شیء در Lightmap. به طور کلی اشیاء Lightmap گرفته شده ی پویا باید دقیقا به یک Lightmap مشترک با پارامترهای یکسان اشاره کنند تا در یک دسته بندی قرار گیرند. اشیاء با Shader هایی که قابلیت Multi-pass دارند سبب شکسته شدن دسته بندی می شوند. تقریبا تمام Shader های یونیتی در حالت Forward Rendering چندین منبع نوری را بطور موثر و در حالات مختلف پشتیبانی می کنند ولی درخواست ترسیم برای پردازش نور در هر پیکسل در دسته بندی قرار نمی گیرد. در روش رندرینگ Legacy Deferred دسته بندی پویا غیر فعال است زیرا هر شیء دوبار ترسیم می شود. دسته بندی پویا به وسیله انتقال کلیه رئوس صحنه به CPU کار می کند در اینصورت این تنها زمانی یک مزیت است که دسته بندی کوچکتری ایجاد شود. منابع مورد نیاز یک درخواست ترسیم به فاکتورهای زیادی وابسته است. در درجه اول API گرافیکی در نظر گرفته می شود. برای مثال بر روی کنسول ها یا API های مدرن شبیه Apple Metal، سربار درخواست ترسیم بطور کلی بسیار کمتر است و اغلب دسته بندی های پویا به هیچ وجه نمی توانند یک مزیت باشند.

۳–۵–۲– دسته بندی پویا (سیستم های ذره ای ، رندرهای خطی ، رندر دنباله دارها): برای اجزایی که دارای هندسه بوده و یونیتی آنها را به صورت پویا تولید می کند، دسته بندی پویا در مقایسه با اشیاء دیگر متفاوت عمل می کند. در موتور یونیتی برای انواع رندرهای سازگار، تمام محتوای قابل دسته بندی در یک بافر Vertex بزرگ ایجاد می شود و سیستم رندر وضعیت ماده را برای بسته تنظیم می کند سپس یونیتی، بافر Vertex را به دستگاه گرافیکی متصل کرده، جایگاه رندر هر یک از بسته ها را در آن بروز رسانی می کند و یک درخواست ترسیم جدید صادر می کند. زمانی که می خواهیم هزینه ی فراخوانی ترسیم در یک دستگاه گرافیکی را اندازه گیری کنیم، کندترین قسمت رندر یک شیء تنظیم وضعیت متریال است و درقیاس با آن، ارسال درخواست ترسیم یک متریال با Offset متفاوت در یک بافر Vertex مشترک سریع تر است. این رویکرد بسیار شبیه ارسال درخواست ترسیم یونیتی در حالت دسته بندی ایستا است.

۳–۵–۳– دسته بندی ایستا: دسته بندی ایستا به موتور اجازه می دهد تا درخواست ترسیم برای احجام هندسی در هر اندازه که باشند را به شرطی که از ماده یکسانی استفاده کرده و حرکت نیز نکند، کاهش دهد. این کار اغلب کارآمدتر از دسته بندی پویا است و محاسبات رئوس را به CPU انتقال نمی دهد، اما از حافظه ی بیشتری استفاده می کند. برای اینکه بتوانید از دسته

بندی ایستا استفاده کنید ، باید دقیقا اشیائی که در محیط ساکن هستند و در بازی حرکت، چرخش یا تغییر مقیاس ندارند را مشخص کنید. استفاده از دسته بندی ایستا به حافظه اضافی برای ذخیره اشیاء ترکیبی نیاز دارد. اگر چندین شیء قبل از دسته بندی ایستا، هندسه یکسانی داشته باشند ، یک نسخه از هندسه در ازای هر شیء در ویرایشگر یا در زمان اجرا ایجاد می شود که ممکن است همیشه ایده خوبی نباشد. بعضی اوقات برای جلوگیری از بجا ماندن ردپای اطلاعات در حافظه، مجبورید بهبود عملکرد رندر را با اجتناب از دسته بندی ایستا برای برخی از اشیاء حاصل کنید. به عنوان مثال ، ایستا کردن درختان در سطح جنگل انبوه می تواند تأثیر جدی در حافظه داشته باشد. بطور کلی روش عملکرد دسته بندی ایستا، انتقال اشیاء ایستا به یک فضای سه بعدی کلی است که در آنجا رئوس مشترک و شاخص بافر برای آنها ساخته می شود. اگر شما گزینه ی بهینه سازی اطلاعات اشیاء را فعال کرده باشید یونیتی تمام رئوس اجزائی که با هیچ نوعی از Shader بکار گرفته نشده را حذف می کند. برای این کار از برخی کلید واژه ها استفاده می کند. به عنوان مثال اگر یونیتی در یک Shader کلمه ی Lightmap_ON را پیدا نکند در دسته بندی Lightmap UVs آن شیء را حذف می کند سپس برای نمایش اشیاء در همان بسته از یک سری درخواست ترسیم ساده که تقریبا هیچ تفاوتی با یکدیگر ندارند استفاده می کند. شاخص دسته بندی در اکثر پلت فرم ها 64k رأس بوده ولی در OpenGLES این شاخص 48k رأس و در macOS به 32k کاهش میابد. در حال حاضر ، فقط Mesh Renderer ها، Trail Renderer، Line Renderer، سیستم های ذره ای و Sprite Renderer قابل دسته بندی هستند. به این معنی که Skin Mesh ها و پارچه ها و یا سایر اجزاء رندرگیری قابل دسته بندی نیستند. رندرها فقط با هم دسته های خود قابل دسته بندی هستند. همچنین در Shaders های نیمه شفاف معمولاً اشیاء به ترتیب عقب ترین به جلوترین ارائه می شوند. یونیتی ابتدا اشیاء را به این ترتیب چینش می کند و سپس سعی می کند آنها را دسته بندی کند، اما از آنجا که ترتیب نمایش باید کاملاً رعایت شود، ممکن است دسته بندی کمتری نسبت به اشیاء غیرشفاف بدست آید. دسته بندی اشیاء نزدیک به یکدیگر می تواند جایگزین بسیار خوبی برای ترسیم بسته ها باشد. به عنوان مثال، منطقی است که یک کمد ثابت با تعدادی کشو به وسیله یک برنامه مدل سازی سه بعدی فقط در یک شیء واحد ساخته، و یا با استفاده از Mesh.CombineMeshes ترکیب شود.

۳–۶– انسداد درخواست ترسیم: یکی از مهم ترین و کلیدی ترین امکانات موتورهای بازی سازی امکان انسداد فراخوانی ترسیم است که این کار در بخش Occlusion Culling انجام می شود. این ابزار کمک می کند تا اجسامی که در کادر رندر قرار ندارند و یا هیچ قسمتی از هندسه ی آنها دیده نمی شود در لیست درخواست ترسیم قرار نگیرند تا زمانی که در فاصله تعیین شده و در کادر رندر قرار گیرند. تنظیمات پیش فرض آن در موتور یونیتی این ضمانت را می دهد که اگر اندازه اجسام با تنظیمات پایه این بخش سازگار و همسان باشند به سریع ترین حالت ممکن محاسبه و نتیجه ی فراخوانی ترسیم نیز عالی باشد. برای این کار باید پارامتر static اجسام ثابت در صحنه را فعال کرده و با در نظر گرفتن واحد اندازه ی هر مسدود کننده Occluder صحنه را چینش کرد سپس با Bake کردن آن محیط به تکه های فرضی جدا سازی می شود و داده ی انسداد را تشکیل می دهد.

۳–۷– Quality parameters: یونیتی به شما امکان تنظیم انواع سطح کیفیت گرافیکی، جهت مقاصد سخت افزاری و سیستم عاملی مختلف را می دهد. به طور کلی، کیفیت برای فریم ریت هزینه بردار است. بنابراین ممکن است که بالاترین

کیفیت را در دستگاه های تلفن همراه یا سخت افزارهای قدیمی نداشته باشید زیرا این امر تأثیرات مخربی روی گیم پلی بازی دارد. شما می توانید با انتخاب سیستم عامل هدف، یکی از کیفیت های پیش فرض در یونیتی را انتخاب کرده و یا هر یک را متناسب با اهداف کیفیتی خود سفارشی سازی کنید. همچنین می توانید تنظیمات خود را به عنوان یک ساختار سفارشی برای خود ذخیره کنید. پارامترهای مهمی در این بخش وجود دارد که هم بر روی مصرف منابع سخت افزاری تاثیر می گذارد و هم بر روی حجم بسته ی نهایی. از جمله پارامتری که بر روی حجم نهایی بازی تاثیر می گذارد Texture Quality است که به حالت های کامل، نیمه، یک چهارم و یک هشتم می توان تنظیم کرد. این عمل نه تنها بر روی مصرف منابع سخت افزاری تاثیر می گذارد بلکه حجم بسته نهایی را نیز کاهش می دهد. گزینه ی Anti-aliasing که می تواند حاشیه بافت ها را نرم تر کند و از ایجاد لبه های تیز در بافت جلوگیری کند در سرعت رندر تاثیر مهمی دارد. پردازش انعکاس های بهنگام در سرعت رندر تاثیر گذار است و پیش نهاد می شود تا جای ممکن از تعداد کمتری از Reflection Probes استفاده شود و در صورت استفاده کیفیت تصویر آنها را در حداقل نیاز حفظ کنید. در تنظیمات Quality می توانید به راحتی نمایش انعکاس های بهنگام را فعال و یا غیر فعال کنید.

۳–۸– برنامه نویسی: در این بخش نحوه بهینه سازی اسکریپت ها و همچنین دلایل عملکرد بهینه سازی ها و استفاده از آنها برای موقعیت های خاص را توضیح خواهیم داد. هیچ چیزی به عنوان لیستی از موارد برای بررسی وجود ندارد که اطمینان حاصل کنید که پروژه بدون مشکل اجرا می شود. برای بهینه سازی یک پروژه ی کند و آهسته ، باید از طریق پروفایلر ایرادات خاصی را پیدا کنید که زمان نامتناسبی را مصرف می کنند. تلاش برای بهینه سازی بدون پروفایلر یا بدون درک دقیق نتایجی که می دهد ، مانند این است که بخواهید با چشم بند بهینه سازی کنید.

۳–۸–۱– مدیریت Garbage Collector: اسکریپت هایی که در یونیتی می نویسید از مدیریت خودکار حافظه استفاده می کنند که تقریباً در مورد تمام زبان های اسکریپت نویسی این کار را انجام می شود. در مقابل، زبانهای سطح پایین مانند C و C++ از تخصیص حافظه دستی استفاده کرده و به برنامه نویس اجازه می دهد مستقیماً از آدرسهای حافظه خوانده و بنویسد، در نتیجه برنامه نویس وظیفه حذف هر شیء ایجاد شده را دارد. به عنوان مثال ، اگر اشیایی را در C++ ایجاد می کنید باید به صورت دستی، حافظه ای را که آنها اشغال کرده بودند را پس از اتمام کار خالی کنید. برای این کار کافی است که کد objectReference = null; را صدا بزنیم ولی در C# این عمل با null کردن انجام نمی شود. بطور مثال اگر یک متغییر GameObject myGameObject; داشته باشید با نوشتن myGameObject = null; حافظه خالی نمی شود زیرا یونیتی مجبور است برای ترسیم، بروز رسانی و غیره مراجع خود را حفظ کند. برای این کار باید از Destroy(myGameObject); استفاده کنید.

۳–۸–۲–توابع تکرار پذیر خودکار: استفاده از عملیات های سنگین مانند حلقه های تو در تو و بزرگ، فراخوانی توابع بازگشتی پیچیده، محاسبات ریاضیاتی سنگین در توابع تکرار پذیر خودکار مانند Update، FixedUpdate، OnGUI و غیر بسیار در عمکرد CPU و آهسته شدن بازی تاثیر دارد. اگر در بازی از یک شیء با تکرار بالا دارید که روی آن یک اسکریپت وجود دارد تابعی مانند Update حتی اگر خالی هم باشد بر عملکرد CPU تاثیر زیادی می گذارد. پس حذف توابع غیر ضروری از روی کدها بسیار اهمیت دارد.

۴– تحلیل و اشکال زدایی به وسیله Profiler

هنگامی که با موتور یونیتی در حال توسعه ی یک بازی هستید، تحلیل و اشکال زدایی آن تا رسیدن به یک خروجی مطمئن و بهینه شده که به درستی در دستگاه های مورد نظر اجرا شود بسیار پر اهمیت است. در یونیتی چندین ابزار وجود دارد که می توانید از آنها برای اندازه گیری عملکرد بازی خود استفاده کنید. یکی از مهم ترین ابزارهای یونیتی Profiler از بخش Analysis است. پروفایلر می تواند وضعیت مصرف پردازشگر اصلی، پردازشگر گرافیکی، حافظه ی اصلی، صدا، رندر و فیزیک سه بعدی یا دوبعدی را در هر فریم تحلیل و بصورت نموداری و جدولی نمایش دهد. در هر یک از گراف ها می توانید عناصر تاثیر گذار بر منابع سخت افزاری را دنبال کنید. بطور مثال در گراف CPU می توانید تاثیر عناصری مانند Rendering، Scripts، Physics، Garbage Collector، VSync و سایر موارد مربوطه را دنبال کرده و یا در گراف GPU می توانید میزان مصرف عناصری مانند مواد مختلف Opaque، Transparent، Shadows/Depth، Deferred Prepass را مشاهده کنید. پروفایلر این امکان را به شما می دهد که در هر فریمی، بازی را متوقف کنید و ببینید که چه چیزی در بازی در حال مصرف منابع سخت افزاری شما است و با بررسی آن می توانید دریابید که آیا امکان کاهش مصرف وجود دارد یا خیر.

۵– نتیجه گیری

در ساخت مدل سه بعدی برای بازی های قابل اجرا بر روی کامپیوتر تعداد راس ها را کمتر از 200K بگیرید و در چیدمان صحنه طوری تنظیم کنید که در هر فریم بیشتر ۳ میلیون راس رندر نشود (بسته به GPU مورد نظر). اگر از شیدرهای از پیش ساخته شده ی یونیتی استفاده می کنید سعی کنید از مجموعه ی شیدر های Mobile و یا Unlit انتخاب کنید. این شیدر ها در پلت فرم های غیر موبایل نیز بخوبی کار می کنند. تعداد مواد مختلف در هر صحنه را کم نگه دارید و تا آنجا که ممکن است مواد را بین اشیا مختلف به اشتراک بگذارید. ویژگی Static را روی یک اشیاء غیر متحرک تنظیم کنید تا بهینه سازی های داخلی مانند دسته بندی استاتیک امکان پذیر شود. فقط یک نور جهت دار(Directional) مؤثر بر مش ها در صحنه داشته باشید. ترجیحا و تا جای ممکن از نور های Bake شده بجای نورهای پویا استفاده کنید. از فرمت های فشرده برای بافت ها استفاده کرده و تا جای ممکن از بافت ها 16-bit بجای بافت های 32-bit استفاده کنید. تا جای ممکن از امکان مه (fog) استفاده نکنید. از Occlusion Culling برای کاهش درخواست ترسیم و نادیده گرفتن اشیاء خارج از محدوده ی رندر استفاده کنید. از Sky box ها برای ایجاد دورنما بصورت جعلی استفاده کنید. از pixel shader ها و یا بافت ها ترکیبی بجای روش multi-pass استفاده کنید. تا جای ممکن از متغییر های نیمه دقیق استفاده کنید. استفاده از عملیات ریاضی پیچیده مانند pow ، sin و cos را در پیکسل شیدرها به حداقل برسانید. در چیدمان صحنه دقت کنید که تا جای ممکن از تعداد بافت های زیاد متمرکز شده جلوگیری کنید.

۶– مراجع:

1. Tiffany Barnes, Heather Richter, Eve Powell, Amanda Chaffin, and Alex Godwin. Game2learn: Building cs1 learning games for retention. In Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE '07, pages 121--125, New York, NY, USA, 2007. ACM.
2. Kajal Claypool and Mark Claypool. Teaching software engineering through game design. In Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE '05, pages 123--127, New York, NY, USA, 2005. ACM.
3. Quincy Brown, Frank Lee, and Suzanne Alejandre. Emphasizing soft skills and team development in an educational digital game design course. In Proceedings of the 4th International Conference on Foundations of Digital Games, FDG '09, pages 240--247, New York, NY, USA, 2009. ACM.

4. Daniel V. de Macedo and Maria Andréia Formico Rodrigues. Experiences with rapid mobile game development using unity engine. Comput. Entertain., 9(3):14:1--14:12, November 2011.

5. Unity Technologies. Unity - game engine. http://docsunity3d.com/. Accessed: 2021-01-05.
6. R. Coleman, M. Krembs, A. Labouseur, and J. Weir. Game design & programming concentration within the computer science curriculum. In Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '05, pages 545--550, New York, NY, USA, 2005. ACM.
7. P. E. Dickson. Using unity to teach game development: When you've never written a game. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '15, pages 75--80, New York, NY, USA, 2015. ACM.
8. Wang Wei-chen, Jiang Xiaohong. Optimizing Real-Time Performance of 3D Virtual Mining Environment with MultiGen Creator. CADDM. 2004. 14(1). 61-69.