روشی نیمه‌خودکار برای یافتن مشکلات گرافیکی بازی‌های رایانه‌ای با استفاده از رایانش ابری

محمدرضا تأثیری[۱]، محمد امین فضلی[۱] و جعفر حبیبی[۱]

[۱] دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف

چکیده

یکی از مشکلات در توسعه‌ی بازی‌های رایانه‌ای، عملکرد نادرست بازی‌ها در محیط‌های اجرایی متفاوت است. بازی‌ها معمولا در محیط توسعه بسیار خوب عمل می‌کنند ولی در رایانه‌های شخصی کاربران با مشکلات زیادی مواجه می‌شوند. در این مقاله، روش نیمه خودکاری را ارائه خواهیم کرد که با استفاده از بستر موجود رایانش ابری برای اجرای بازی‌های رایانه‌ای، مشکلات گرافیکی احتمالی را بر روی محیط‌های اجرایی مختلف، کشف و گزارش می‌کند. با این ابزار توسعه‌دهندگان بازی‌های رایانه‌ای به راحتی می‌توانند نحوه‌ی عملکرد بازی خود را بر روی محیط‌های اجرایی متفاوت مورد بررسی قرار دهند.

کلمات کلیدی

بازی‌های رایانه‌ای، آزمون بازی‌های رایانه‌ای،آزمون نیمه خودکار، رایانش ابری

۱ مقدمه

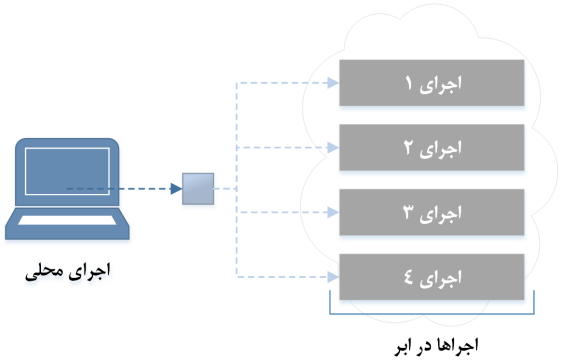
امروزه صنعت بازی‌سازی یکی از پردرآمدترین صنعت‌ها در جهان است. رشد این صنعت در چند سال اخیر بسیار افزایش یافته است به طوری که در سال ۲۰۱۵ فروش بازی‌های رایانه‌ای در ایالات متحده آمریکا به ۲۳.۵ میلیارد دلار رسیده است [۱]. روند تولید بازی‌های رایانه‌ای با سایر نرم‌افزارها متفاوت است و اغلب آزمون این محصولات نرم‌افزاری با مشکلات زیادی همراه است. به طور معمول، تیم‌های مختلف توسعه‌ی بازی با توجه به نوع بازی در حال ساخت، هر کدام روش خاصی را برای آزمون بازی اتخاذ می‌کنند. از مشکلات رایج پس از انتشار یک بازی، ناسازگار بودن آن با رایانه‌های شخصی کاربران بازی است. در چند سال اخیر به‌روزرسانی بازی در روزها وهفته‌های اولیه‌ی پس از انتشار بازی برای رفع مشکلات کوچک و سازگاری با محیط‌های اجرایی مختلف، تبدیل به یکی از بزرگ‌ترین معضلات توسعه‌دهندگان بازی شده است. برای مثال مشکلات شدید فنی نسخه‌ی رایانه‌ی شخصی بازی Batman: Arkham Knight در سال ۲۰۱۵ منجر به توقف فروش این بازی شد [۲].

از طرف دیگر توسعه‌ی رایانش ابری این امکان را به‌وجود آورده است که بازی‌ها بر بستر رایانش ابری به اجرا در آمده و خروجی تصویر بازی به رایانه و یا تلفن همراه کاربران ارسال شود. قابلیت اجرای بازی در ابر فرصت خوبی برای آزمون بازی را در اختیار قرار می‌دهد. تمرکز ما در این مقاله بر روی کشف و شناسایی مشکلات کارایی و گرافیکی احتمالی یک بازی در محیط‌های اجرایی مختلف است و این کار با استفاده از اجرای مجدد قطعی و خودکار یک بازی بر روی محیط‌های اجرایی که توسط رایانش ابری فراهم آورده شده‌اند، انجام می‌شود.

۲ پژوهش‌های پیشین

پژوهش‌هایی که در حوزه‌ی آزمون بازی‌های رایانه‌ای که سابق بر این انجام شده است را در این بخش معرفی خواهیم کرد.

در [۳] چند روش ابتکاری برای مشخص کردن تأثیر ارزیابی‌های قابلیت استفاده در بازی‌های رایانه‌ای معرفی شده است. در [۴] و [۵] تکنیک‌هایی برای آزمون قابلیت استفاده یک بازی ارائه شده است. در [۶] روشی برای انجام آزمون دود در بازی‌ها ارائه شده است. در [۷] روش آزمون اسکریپت محور برای آزمون مستقل از سکو بازی‌ها معرفی شده است. در [۸] مجموعه روش‌های خودکاری برای آزمون بازی معرفی شده است که باعث یک شرکت کم‌تر به آزمون بتا متکی باشد. رویکردی مبتنی بر حالت برای آزمون نمونه‌های اولیه بازی در [۹] معرفی شده است. در [۱۰] با استفاده از الگوریتم‌های ژنتیک روشی ارائه شده است تا رفتارهای نامطلوب بازی کشف شوند. در [۱۱] با استفاده از هوش مصنوعی و الگوریتم‌های بینایی ماشین روش برای آزمون بازی معرفی شده است. در [۱۲] روش سناریو محور جعبه سیاهی که با استفاده از گرامر و نقشه‌ی بازی می‌توان بازی را آزمون کرد، ارائه شده است. در [۱۳] روشی برای آزمون خودکار بازی‌های آنلان با استفاده از تعداد بسیار زیادی بازی‌کن مجازی معرفی شده است. در [۱۴]



شکل ۱: معماری سیستم

روشی مدل محور برای آزمون خودکار بازی‌های سکو معرفی شده است. روشی برای خودکارسازی آزمون بازگشت بازی‌ها در [۱۵] ارائه شده است. در [۱۶] روشی خودکار برای پیدا کردن مشکلات در بازی‌ها ارائه شده است. در این روش حالات درونی بازی کنترل و ضبط شده و با استفاده از LTL-FO$^+$ که تعمیمی از منطق زمانی خطی است تغییر حالت‌های غیرمجاز شناسایی و گزارش می‌شوند.

۳ معماری سیستم

در این مقاله ما راه‌کاری را ارائه می‌کنیم که با استفاده از زیرساخت‌های موجود رایانش ابری برای بازی‌های رایانه‌ای، بتوان تعداد زیادی نمونه از یک بازی را در محیط‌های اجرایی متفاوت به اجرا در آورده و با مقایسه هر یک از این اجراها با یک اجرای مرجع در سطح فراخوان‌های گرافیکی، به دنبال مشکلات احتمالی گشت. از عمده مشکلات بازی‌های رایانه‌ای، عدم اجرای یکسان بر روی محیط‌های اجرایی متفاوت است. روشی که ما ارائه می‌دهیم به توسعه‌دهنده‌ی بازی این امکان را می‌دهد که با هر بار آزمون بازی بر روی رایانه‌ی محلی خود، رفتار بازی را در چندین محیط اجرایی مختلف زیر نظر بگیرد. سیستم طراحی شده توسط ما، اجراهای مختلف را با اجرای محلی در سطح فراخوان‌های رسم مقایسه کرده و موارد مشکوک به خطا را گزارش می‌کند. برای آن‌که مقایسه کاملا بین فریم‌های یکسانی انجام شود، روشی برای قطعی کردن رفتار بازی در تمامی اجراها ارائه شده است تا به صورت کاملا دقیق نتایج اجراهای مجدد در بستر رایانش ابری همان چیزی باشد که توسعه‌دهنده‌ی بازی در رایانه‌ی محلی خود می‌بیند. در شکل ۱ معماری سیستم طراحی شده را می‌توان دید.

۳-۱ فرضیات و نیازمندی‌ها برای آزمون

در این قسمت فرض‌ها و نیازمندها در طراحی سیستم بیان شده است. اولین و مهم‌ترین فرض در نظر گرفته‌شده در طراحی سیستم، ثابت بودن ترتیب کلی فراخوان‌های رسم تولید شده توسط موتوربازی‌سازی است. با توجه به مشاهدات ما و الگوریتم‌های رندر در موتورهای بازی‌سازی، در اکثر بازی‌ها این فرض برقرار است. یکی دیگر از فرض‌ها یکسان بودن وضوح تصویر در اجراهای مختلف است. از آن‌جا که تصاویر تولید شده توسط کارت گرافیک، ورودی الگوریتم‌های مقایسه هستند؛ متفاوت بودن وضوح این تصاویر باعث ایجاد خطا در گزارش‌ها و نتایج می‌شود. در روند آزمون، نیاز به یک رایانه‌ی مرکزی و یک بازی‌کن است تا بازی را به اجرا درآورد. پس از اجرا، به صورت خودکار سایر اجراها را با اجرای محلی هم‌گام می‌شوند. به طبع در صورتی که بازی دارای مامورین هوش مصنوعی باشد، می‌توان از این مامورین برای بازی استفاده کرد. سیستم طراحی شده در حال حاضر قابلیت آزمون بازی‌ها با رابط گرافیکی Direct3D 11 را دارا می‌باشد؛ اما روش‌های ارائه شده به سایر رابط‌های برنامه‌نویسی گرافیکی از جمله OpenGL و Vulkan قابل تعمیم است.

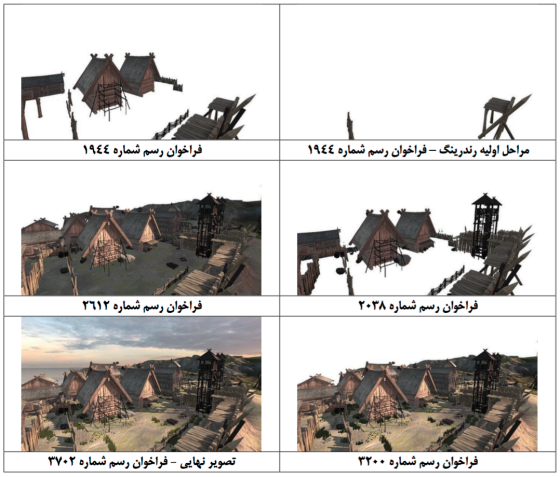
۴ مقایسه دو اجرا

بازی‌های رایانه‌ای دسته‌ای از نرم‌افزار است که خروجی آن‌ها تصاویر گرافیکی می‌باشد که بر روی صفحه‌ی نمایش‌گر نشان داده می‌شود. مشکلاتی که ما در این مقاله به دنبال آن هستیم، مشکلات گرافیکی هستند که باعث می‌شود تصویر نهایی نشان داده شده به کاربر مخالف انتظار سازنده‌ی بازی باشد. از عمده دلیل‌های به‌وجود آمدن این ناسازگاری‌ها، استفاده از توابع و عملکردهایی هست که استفاده از آن‌ها فقط بر روی یکسری از کارت‌های گرافیک و یا یک نسخه‌ی خاص از یک سیستم عامل ممکن است. راه‌کار ابتدایی برای مشخص کردن فریم‌های دارای مشکل گرافیکی، مقایسه تصویر نهایی تولید شده (مقایسه Back Buffer‌ها) با یکدیگر است. متاسفانه استفاده تنها از تصویر تولید شده و نادیده گرفتن سایر اطلاعات رندرینگ منجر به دشوارتر شدن مسئله و کاهش دقت نتایج می‌شود. ابتدا به چند مشکل مربوط به این راه‌کار اشاره کرده و سپس راه‌کاری برای حل این مشکلات مطرح خواهیم نمود.

یکی از مشکلات تاثیر اشیایی که در تصویر نهایی به صورت مستقیم قابل رویت نیستند؛ اما بر روی تصویر نهایی تأثیر می‌گذارند. این دسته از مشکلات در نورپردازی خود را به خوبی نشان می‌دهند. مسئله‌ی دیگر استفاده از روش‌هایی مانند Multiple Render Target است. با استفاده از Multiple Render Target برنامه‌نویس قادر است به‌طور هم‌زمان تصاویری را در چند Render Target get Texture ایجاد کند و با استفاده از آن‌ها تصویر نهایی را تولید کند. به علت وجود چنین مشکلاتی، تشخیص منبع مشکل گرافیکی تنها با استفاده از تصویر نهایی کاری دشوار خواهد بود. راه‌حل ساده برای حل این مشکلات؛ استفاده از فراخوان‌های گرافیکی بازی در زمان اجرا است. در بخش بعد در مورد نحوه‌ی رندر شدن یک فریم توضیح خواهیم داد.

۴-۱ نحوه‌ی رندر شدن یک فریم

برای رندر شدن یک صحنه و تولید تصاویر نهایی لازم است داده‌هایی شامل مدل‌های سه‌بعدی، بافت‌ها، ماتریال و ... از حافظه اصلی به حافظه‌ی کارت گرافیک منتقل شوند. پس از این انتقال، پردازش‌گر مرکزی توسط فراخوان‌های رسم به کارت گرافیک دستور می‌دهد تا یک شی را رندر نماید. با توجه به پیچیدگی یک صحنه، تعداد فراخوان‌های رسم مورد نیاز برای تولید تصویر نهایی متفاوت است و در برخی موارد این تعداد به ۱۰ هزار نیز می‌رسد. در شکل ۲ روند رندر شدن یک صحنه که حدود ۴ هزار فراخوان رسم برای تولید آن نیاز است را می‌توان مشاهده کرد. نکته‌ی مهم در مورد فراخوان‌های رسم، ترتیب آن‌ها می‌باشد. ترتیب شکل‌گیری فراخوان‌های رسم به الگوریتم‌های رندرینگ و معماری موتوربازی‌سازی مرتبط است. یکی از فرض‌های در طراحی سیستم این است که موتور بازی‌سازی برای فراخوان‌های رسم خود ترتیب ثابتی در نظر می‌گیرد. مشاهدات ما نشان می‌دهد موتورهای بازی‌سازی Unreal Engine، Unity و Glacier 2 ترتیب



شکل ۲: مراحل رندر شدن یک فریم توسط موتور بازی‌سازی.

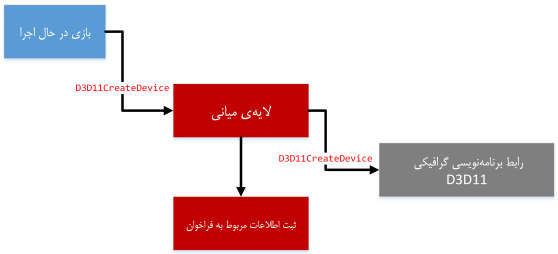
ثابتی برای فراخوان‌های رسم خود در نظر گرفته‌اند.

۴-۲ ذخیره فریم‌ها

برای به‌دست آوردن فراخوان‌های گرافیکی و استفاده از آن‌ها در مقایسه‌ی فریم‌ها، کافی است مانند تمامی ابزارهای اشکال‌زدایی گرافیکی، تمامی فراخوان‌های مربوط به یک فریم را ذخیره کنیم. برای ذخیره کردن فراخوان‌های گرافیکی، کافی است یک لایه بین بازی و رابط برنامه‌نویسی گرافیکی گنجاند و در این لایه، فراخوان‌های گرافیکی را از بازی دریافت، ذخیره و سپس به رابط برنامه‌نویسی گرافیکی حقیقی ارسال نمود. در حقیقت این لایه مفهومی مشابه حملات «مرد میانی»[۱] در امنیت شبکه و رایانه دارد. شکل ۳ نحوه‌ی عملکرد این لایه را نشان می‌دهد. در حوزه‌ی اشکال‌زدایی گرافیکی، ابزارهای زیادی وجود دارند که این لایه را پیاده‌سازی کرده‌اند. vogl [۱۷]، Visual Studio Graphics Tools [۱۸] و RenderDoc [۱۹] نمونه‌هایی از این ابزارها هستند. در این مقاله ما از ابزار RenderDoc برای ذخیره‌ی فراخوان‌های توابع گرافیکی استفاده کرده‌ایم.

۴-۳ تطبیق فراخوان‌ها

در روش ما مقایسه بین فریم‌های حاصل از اجرا بر روی بستر رایانش ابری و فریم مرجع در سطح فرخوان‌های رسم انجام می‌شود. فریم مرجع فریمی است که از رایانه‌ی مرکزی تهیه شده و توسعه‌دهنده‌ی بازی آن را به عنوان خروجی مورد انتظار بازی می‌داند. پیش از انجام هرگونه مقایسه، ابتدا باید فرخوان‌های رسم را بین اجراها تطبیق داد. همان‌طور که در بخش قبل گفته شد ترتیب فراخوان‌های رسم در یک صحنه یکسان فرض شده است. ممکن است به دلایلی یک فراخوان رسم در فریم مرجع باشد ولی در برخی از اجراها اتفاق نیفتد. نام این دسته از فراخوان‌ها را **فراخوان گمشده** می‌گذاریم. یک فراخوان گم‌شده می‌تواند نشان‌گر یک مشکل گرافیکی باشد ولی این مسئله همواره صحیح نیست. برای مثال اعمال تغییرات در تنظیمات برای کاهش جزئیات باعث به‌وجود آمدن فراخوان‌های گم‌شده می‌شود اما این مسئله مشکل گرافیکی نیست. هرگاه در یکی از اجراها یک فراخوان رسم وجود داشته باشد که فراخوان متناظری برایش در اجرا اصلی وجود نداشته باشد، به آن **فراخوان یتیم** گوییم. یکی از دلایل به‌وجود آمدن فراخوان‌های یتیم، الگوریتم‌های تغییر سطوح جزئیات است. این الگوریتم‌ها با توجه به منابع محیط



شکل ۳: ذخیره و بازاجرای فراخوان‌های گرافیکی.

اجرایی، به صورت خودکار جزئیات برخی از اشیا را کاهش داده و یا برخی از اشیا را از صحنه حذف می‌نماید. در زمانی که الگوریتم سطوح جزئیات یک شی را کاملا با شی دیگر تعویض کند؛ با توجه به ساختار هندسی دو شی، تطبیق فراخوان‌ها رسم دو شی در دو اجرا دشوار و در برخی مواقع غیر ممکن است.

برای تطبیق فراخوان‌ها رسم بین دو اجرا می‌توان از دستورات نشان‌گر رویداد رابط‌های برنامه‌نویسی گرافیکی، مانند `D3DPERF_SetMarker`، استفاده کرده و اطلاعات اضافی قبل از هر فرخوان رسم ثبت نمود. استفاده از این روش در همه‌ی موتورهای بازی‌سازی ساده و یا ممکن نیست، از این رو ما روش خودکاری را برای پیدا کردن یک تطبیق ماکسیمال بین فراخوان‌های رسم ارائه می‌کنیم. برای انجام این کار از پارامترهای فراخوان رسم و اطلاعات رئوس اشیا استفاده خواهیم کرد.

با یک‌بار پیمایش فراخوان‌های رسم، آن‌ها را در یک جدول درهم‌سازی درج می‌کنیم. تعداد رئوس شی رسم شده یک فراخوان را به عنوان کلید جدول درهم‌سازی در نظر می‌گیرم. فراخوان‌هایی که به یک خانه از جدول درهم‌سازی نگاشت می‌شوند را به ترتیب در یک آرایه درج خواهیم کرد. تطبیق این فراخوان‌ها با توجه به ترتیب درج آن‌ها و با استفاده از ساختار هندسی شی، ممکن است.

۴-۴ مقایسه فریم‌ها

برای محاسبه‌ی میزان تفاوت دو تصویر مقادیر Mean square error (فرمول ۱)، Kullback–Leibler divergence (KL) (فرمول ۲) و Structural similarity index metric (SSIM) (فرمول ۳) را محاسبه خواهیم کرد. مقدار MSE عددی مثبت است که هر چه به صفر نزدیک‌تر باشد نشان‌گر نزدیک بودن دو تصویر است. مقدار KL divergence می‌تواند عددی مثبت و یا منفی باشد و هر چه این مقدار به صفر نزدیک‌تر باشد نشان‌گر نزدیک بودن دو تصویر است. از طرف دیگر مقدار SSIM عددی بین ۱- و ۱ است که هر چه این عدد به ۱ نزدیک‌تر شود دو تصویر شباهت بیش‌تر به هم دارند. برای هر فراخوان رسم تطبیق داده شده، پیکسل‌های تحت تأثیر از آن فرخوان را در هر یک از Render Targetها محاسبه خواهیم کرد و مقادیر SSIM، MSE و KL divergence برای هر یک از آن‌ها محاسبه خواهیم کرد. برای فراخوان‌ها رسم گم شده و یتیم که نتوانسته‌ایم برای آن‌ها تطبیقی پیدا نماییم، از پیکسل‌های مرتبط با این فراخوان‌ها در تصویر نهایی استفاده کرده و مقادیر مذکور را به‌دست خواهیم آورد.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(O_i - R_i)^2 \qquad (۱)$$

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\ \log \frac{P(i)}{Q(i)}. \qquad (۲)$$

$$\mathrm{SSIM}(O,R) = \frac{(2\mu_O\mu_R + c_1)(2\sigma_{OR} + c_2)}{(\mu_O^2 + \mu_R^2 + c_1)(\sigma_O^2 + \sigma_R^2 + c_2)} \qquad (۳)$$

۵-۴ ردیابی اشیا در فریم‌های مختلف

یک شی می‌تواند در یک فریم چند بار ظاهر شده و یا در فریم‌های مختلفی حضور داشته باشد. با شناسایی یک شی در فریم‌ها مختلف و محاسبه‌ی مقادیر SSIM، MSE و KL divergence می‌تواند اطلاعات بهتری از نحوه‌ی رندر شدن یک شی در حالات مختلف به‌دست آورد. برای ردیابی اشیا کافی است از مختصات رئوس در فراخوان‌های رسم استفاده نمود.

۵ اجرای مجدد قطعی

برای مقایسه فریم‌ها، یکسان بودن خروجی گرافیکی بازی در اجراها الزامی است. بسته به موتوربازی‌سازی و نوع بازی، ممکن است قابلیت اجرای مجدد پیاده‌سازی شده باشد. ما روشی تحت عنوان «اجرای مجدد قطعی» که ایده‌ی آن برگرفته از [۲۰] را ارائه خواهیم کرد که با تغییرات کمی در کدهای موتوربازی‌سازی می‌توان آن را پیاده‌سازی کرد.

در صورتی که ورودی‌ها بازی‌کن از طریق رایانه‌ی مرکزی به سایر اجراها در فریم‌ها درست اعمال شود و عوامل غیرقطعی کننده‌ی خروجی بازی حذف شود، می‌توان تمامی اجراهای یک بازی را کاملا هم‌گام کرد. اگر تاخیر ارسال ورودی‌ها از رایانه‌ی مرکزی به ابر فقط به اندازه‌ی یک فریم باشد؛ اثر فشرده‌شدن یک کلید در اجرای محلی بر روی فریم شماره n و در سایر اجراها در فریم $n+1$ اعمال می‌شود، و این اتفاق باعث متفاوت شدن خروجی‌ها خواهد شد. بدیهی است در صورتی که بیش‌تر بودن زمان تاخیر و متفاوت بودن زمان تولید فریم‌ها در اجراهای متفاوت، اختلاف خروجی‌ها بیش‌تر خواهد شد.

در [۲۰] عوامل غیر قطعی کننده‌ی یک بازی بررسی شده و تابع `GetSystemTime`، مولدهای اعداد تصادفی و شدت صدای موسیقی بازی به عنوان این عوامل معرفی شده‌اند. برای حل مشکل اعداد تصادفی کافی است مقدار دانه در این الگوریتم‌ها یکسان شوند. به طور معمول از زمان سیستم به عنوان دانه در مولد استفاده می‌شود. در برخی از بازی‌ها، شدت صدا و شدت نور در محیط بازی رابطه‌ی مستقیم دارد. در صورتی که در یک بازی، موسیقی بر نحوه‌ی رندر شدن صحنه تاثیر می‌گذارد، می‌توان از علائم اجرایی برای همگام قطعی‌سازی بافرهای صدا استفاده کرد. در بازی‌های آزمون‌شده توسط ما موسیقی تأثیری در رندرینگ نداشته و پیاده‌سازی این قسمت در سیستم ما انجام نشده است. مسئله‌ی دیگر `DeltaTime` یا فاصله‌ی زمان رندر شدن دو فریم متوالی است. این مقدار که با dt نیز نمایش داده می‌شود در محاسبات موتور فیزیک بازی نقش مهمی دارد. با توجه به خطای ذخیره‌سازی اعداد اعشاری، ممکن است دو عبارت $vdt+vdt$ و $2vdt$ با هم اختلاف اندکی داشته باشند؛ و انجام دو بار شبیه‌سازی فیزیک با vdt با یک‌بار شبیه‌سازی با $2vdt$ منجر به نتایج متفاوتی شود.

برای حل مشکلات ذکر شده این‌گونه عمل خواهیم کرد: زمانی که بازی‌کن در رایانه‌ی مرکزی شروع به بازی می‌کند، تمامی فریم‌ها به همراه مهر زمانی و ورودی‌های آن فریم ثبت و در جدولی به نام جدول رویدادها ذخیره می‌شود. اجراها بر بستر رایانش ابری هر کدام در بازه‌های زمانی ثابت جدول رویدادها را از رایانه‌ی مرکزی دریافت کرده و بر طبق آن عمل خواهند کرد. از آن‌جا که زمان رندر شدن فریم در اجراها متفاوت است باید مشکل dt را حل نمود. به منظور هم‌گام‌سازی اجراها، مقدار dt در اجراها در ابر با این مقدار در رایانه‌ی مرکزی جایگزین می‌شود. روند اجرا تا زمانی که سطرهای پردازش نشده‌ای در جدول رویدادها هست، ادامه پیدا می‌کند. پس از پایان یافتن سطرهای پردازش نشده، تا زمان رسیدن اطلاعات بیش‌تر، یک اجرا متوقف خواهد شد. راه ساده برای جایگزین کردن مقدار dt اعمال تغییر در فراخوانی‌های تابع `GetSystemTime` و یا توابع مشابه برای خواندن ساعت سیستم است. بسته به معماری موتوربازی‌سازی می‌توان در سطوح مختلف این مقدار را جایگزین نمود.

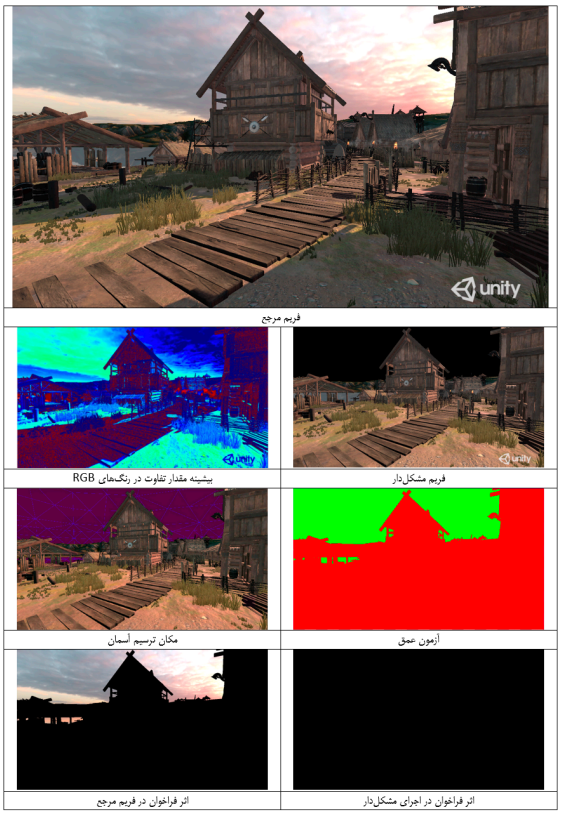
۶ ارزیابی

برای شبیه‌سازی محیط رایانش ابری با استفاده از VGA Passthrough در KVM و QEMU، چهار ماشین مجازی ساخته و به هر کدام یک کارت گرافیک مجزا اختصاص داده شده است. هر کدام از ماشین‌های مجازی دارای۴ هسته‌ی مجزا پردازش‌گر و به آن‌ها ۶ گیگابایت حافظه رم مجزا تخصیص داده شده است. علاوه بر این ماشین‌ها، ماشین مجازی دیگری در VMware ساخته و قابلیت 3D Graphics Acceleration در آن فعال گردیده است. مشخصات دقیق ۵ ماشین مجازی در جدول ۱ قابل رویت است.

جدول ۱: مشخصات محیط اجرایی

نام	توضیحات
پردازش‌گر مرکزی	Dual Xeon E5-2620v3
کارت گرافیک ماشین ۱	NVIDIA GTX 780 Ti
کارت گرافیک ماشین ۲	AMD Radeon R9 270
کارت گرافیک ماشین ۳	NVIDIA GT 740
کارت گرافیک ماشین ۴	AMD Radeon HD6450
کارت گرافیک ماشین ۵	VMware Accelerated 3D Graphics
حافظه رم	4x 8GB DDR4 ECC 2400MHz
حافظه SSD	SAMSUNG 850 EVO 1TB SATA III
هایپروایزر ۱	QEMU + KVM
هایپروایزر ۲	VMware Workstation 12.5 Pro
سیستم عامل میهمان	Microsoft Windows 10.0.14393

در ارزیابی سیستم، از پروژه‌های نمونه‌ی موتوربازی‌سازی Unity استفاده شده است. با اعمال تغییرات اندکی، قابلیت اجرای مجدد قطعی به این پروژه‌ها اضافه شده است. بازی به صورت هم‌زمان در ۵ ماشین مجازی اجرا شده و در زمان‌های مشخص اطلاعات فراخوان‌های رسم از آن‌ها استخراج شده است. در اولین آزمون، ماشین مجازی ۵ قادر به رندر کردن آسمان نبوده و آن را کاملا سیاه نشان داده است. در شکل ۴ می‌توان جزئیات فراخوان رسم آسمان را مشاهده کرد. در ماشین مجازی ۵ فراخوان رسم مربوط به آسمان اتفاق افتاده است ولی تصویر توسط کارت گرافیک تولید نشده است و در نتیجه آسمان سیاه رنگ رندر شده است. در چنین شرایطی که



شکل ۴: مقایسه فریم مرجع و فریم مشکل‌دار

فراخوان رسم اتفاق می‌افتد ولی تصویری تولید نمی‌شود به راحتی می‌توان فراخوان رسم مشکل‌دار را کشف و شناسایی نمود.

برای تولید موارد آزمون بیش‌تر، در یکی از اجراها ماتریالی را حذف کرده و عملیات مقایسه را انجام داده‌ایم. با توجه به مشاهدات ما مقدار SSIM به تنهایی معیار قوی در شناسایی فراخوان‌های مشکل‌دار است. مقدار SSIM در فراخوان‌های مشکل‌دار با اختلاف زیادی ازسایر فراخوان‌ها کم‌تر بوده است و می‌توان معیار خوبی برای دسته‌بندی فراخوان‌ها در نظر گرفته شود. در شکل ۵ رندرهای مختلف یک شی با استفاده از SSIM، MSE و KL divergence با فریم مرجع مقایسه شده‌اند.

۷ نتیجه‌گیری و کارهای آتی

طبق آزمایش‌های انجام شده و نتایج به‌دست آمده توسط ما، می‌توان نتیجه گرفت مقایسه‌ی فراخوان‌های رسم برای پیدا کردن مشکلات گرافیکی روش موثری است. با استفاده از این اطلاعات علاوه بر این‌که می‌توان مشکلات احتمالی گرافیکی را در محیط‌های اجرای مختلف شناسایی کرد؛ می‌توان عامل به‌وجود آورنده را نیز شناسایی کرد.

برای خودکارسازی کامل سیستم می‌توان از سایر الگوریتم‌های پردازش تصویر و یادگیری ماشین استفاده کرده و فراخوان‌ها رسم را به دسته‌ها سالم و خطادار دسته‌بندی نمود.

از کاربردهای دیگر سیستم ارائه شده توسط ما، آزمون بازی‌های تلفن همراه است. در چند سال اخیر بسترهای تلفن همراه در ابر برای مقاصد آزمون نرم‌افزار

شکل ۵: مقایسه رندرها از یک شی

معرفی شده‌اند. از میان آن‌ها می‌توان به AWS Device Farm [۲۱] و Fire-base Test Lab [۲۲] اشاره نمود. توسعه‌دهندگان نرم‌افزارهای موبایل می‌توانند از این بستر برای آزمون نرم‌افزار خود و سازگاری آن با دستگاه‌های تلفن همراه مختلف استفاده کنند. با استفاده از روش ارائه شده در این مقاله می‌توان مشکلات گرافیکی را در تلفن‌های همراه مختلف کشف و گزارش کرد. چالشی که سکوهای تلفن‌های همراه ایجاد می‌کنند، متفاوت بودن اندازه‌ی صفحه‌ی نمایش و به طبع وضوح تصویر است.

مراجع

[1] E. S. Association *et al.*, "Essential facts about the computer and video game industry," 2016.

[2] "Sales of batman: Arkham knight's pc version suspended on steam (update)," 2015. [Online; accessed 20-December-2016].

[3] Y. J. Choi, "Providing novel and useful data for game development using usability expert evaluation and testing," in *Computer Graphics, Imaging and Visualization, 2009. CGIV'09. Sixth International Conference on*, pp.129–132, IEEE, 2009.

[4] N. M. Diah, M. Ismail, S. Ahmad, and M. K. M. Dahari, "Usability testing for educational computer game using observation method," in *Information Retrieval & Knowledge Management,(CAMP), 2010 International Conference on*, pp.157–161, IEEE, 2010.

[5] P. Moreno-Ger, J. Torrente, Y. G. Hsieh, and W. T. Lester, "Usability testing for serious games: Making informed design decisions with user data," *Advances in Human-Computer Interaction*, vol.2012, p.4, 2012.

[6] C. Buhl and F. Gareeboo, "Automated testing: a key factor for success in video game development. case study and lessons learned," in *Proceedings of Pacific NW Software Quality Conferences*, pp.1–15, 2012.

[7] K. Peterson, S. Behunin, and F. Graham, "Automated testing on multiple video game platforms," Feb. 4 2011. US Patent App. 13/020,959.

[8] C. Schaefer, H. Do, and B. M. Slator, "Crushinator: A framework towards game-independent testing," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pp.726–729, IEEE, 2013.

[9] A. M. Smith, M. J. Nelson, and M. Mateas, "Computational support for play testing game sketches.," in *AIIDE*, 2009.

[10] B. Chan, J. Denzinger, D. Gates, K. Loose, and J. Buchanan, "Evolutionary behavior testing of commercial computer games," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol.1, pp.125–132, IEEE, 2004.

[11] A. Nantes, R. Brown, and F. Maire, "A framework for the semi-automatic testing of video games.," in *Artificial Intelligence for Interactive Digital Entertainment (AIIDE), 2008 The Fourth International Conference on*, 2008.

[12] C.-S. Cho, K.-M. Sohn, C.-J. Park, and J.-H. Kang, "Online game testing using scenario-based control of massive virtual users," in *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, vol.2, pp.1676–1680, IEEE, 2010.

[13] Y. Choi, H. Kim, C. Park, and S. Jin, "A case study: Online game testing using massive virtual player," in *Control and Automation, and Energy System Engineering*, pp.296–301, Springer, 2011.

[14] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *Model Driven Engineering Languages and Systems (MODELS), 2015 ACM/IEEE 18th International Conference on*, pp.426–435, IEEE, 2015.

[15] M. Ostrowski and S. Aroudj, "Automated regression testing within video game development," *GSTF Journal on Computing (JoC)*, vol.3, no.2, p.60, 2013.

[16] S. Varvaressos, K. Lavoie, A. B. Massé, S. Gaboury, and S. Hallé, "Automated bug finding in video games: A case study for runtime monitoring," in *Software Testing, Verification and Validation, 2014 IEEE Seventh International Conference on*, pp.143–152, IEEE, 2014.

[17] "vogl, opengl capture / playback debugger.," 2016. [Online; accessed 20-December-2016].

[18] "Visual studio graphics diagnostics - msdn - microsoft," 2016. [Online; accessed 20-December-2016].

[19] "Renderdoc, a stand-alone graphics debugging tool.," 2016. [Online; accessed 20-December-2016].

[20] E. Cuervo, A. Wolman, L. P. Cox, K. Lebeck, A. Razeen, S. Saroiu, and M. Musuvathi, "Kahawai: High-quality mobile gaming using gpu offload," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp.121–135, ACM, 2015.

[21] "Mobile app testing on devices – aws device farm," 2016. [Online; accessed 20-December-2016].

[22] "Firebase test lab for android," 2016. [Online; accessed 20-December-2016].

پانویس‌ها

[1] Man in the Middle Attack