دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

پایان‌نامه کارشناسی

طراحی و پیاده‌سازی سیستم برش اشیاء در محیط گرافیکی سه‌بعدی

نگارنده

امیرحسین بینش

استاد راهنما

دکتر حمیدرضا زرندی

مهر ۱۴۰۰

به نام خدا

دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

تاریخ: مهر ۱۴۰۰

تعهد نامه اصالت اثر

اینجانب **امیرحسین بینش** متعهد می شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است

امیرحسین بینش

امضا

تقدیر و تشکر

سپاس خدایی را که آمرزنده و مهربان است و با فضل بیکرانش، از نیکوکاران در راه خود سپاس‌گزاری می‌کند، حال آن که سزاوار است جنبندگان زمین و آسمان ها بامدادان و شامگاهان او را تسبیح گویند و باز هم ناتوان در ستایش نعمت های بیکرانش، سرگشته و حیران اند. این پایان‌نامه را علی‌رغم اندک دارایی علمی، به تمام کسانی تقدیم می‌کنم که آرزوهای بزرگ در سر می‌پرورانند و در جهت پیشرفت انسانیت قدم برمی‌دارند.

- از پدر و مادر عزیزم که از هیچ دلسوزی دریغ نکرده‌اند و در مشکلات زندگی پشتم ایستاده‌اند و در تصمیم‌گیری‌ها به من اعتماد کردند؛
- از استاد فرزانه و گرامی، دکتر حمیدرضا زرندی که که در این پژوهش با حسن خلق راهنمای بنده بوده‌اند؛
- از استاد گرامی، دکتر حامد فربه که با بزرگواری، بار داوری این پروژه را تقبل فرموده‌اند؛

کمال تشکر و قدردانی را دارم.

امیرحسین بینش

مهر ۱۴۰۰

چکیده

برش اشیاء یکی از مکانیزم‌های پرطرفدار بازی‌های ویدئویی است و مسئله‌ای پیچیده است که سال‌ها پژوهشگران در حال بهبود آن هستند. در این پروژه سعی شده با نیم‌نگاه ساخت یک ابزار همه‌جانبه و قابل گسترش، یک افزونه برش اشیاء با کارایی قابل توجه در محیط سه‌بعدی بر روی موتور بازی‌سازی یونیتی توسعه داده شود.

این پایان‌نامه فرایند برش یک شی سه‌بعدی محدب[1] را در فضای گرافیکی سه‌بعدی توصیف می‌کند. برش توری[2] سه‌بعدی عملیاتی است که یک شی سه‌بعدی را به کمک یک صفحه برش، به دو قسمت مساوی یا نامساوی تقسیم می‌کند. برای پر کردن فضای برش خورده از روش‌های مثلثی‌سازی[3] کمک گرفته شده است. تکنیک‌های بهبود در این پایان‌نامه به همراه یک روش برای مرتب سازی رئوس شی سه‌بعدی پیشنهاد شده است. برای نمایش قدرت ابزار توسعه داده شده نیز، آزمایشات لازم انجام شده و به نتایج قابل توجهی دست پیدا شده است. در نهایت یک بازی برای اثبات کار ارائه گردیده است.

واژه‌های کلیدی:

برش شی سه‌بعدی، مثلثی‌سازی، قشر محدب[4]، هندسه محاسباتی[5]، دستکاری توری[6]

[1] Convex
[2] Mesh
[3] Triangulation
[4] Convex Hull
[5] Computational Geometry
[6] Mesh manipulation

فهرست مطالب

فهرست اشکال و نمودارها

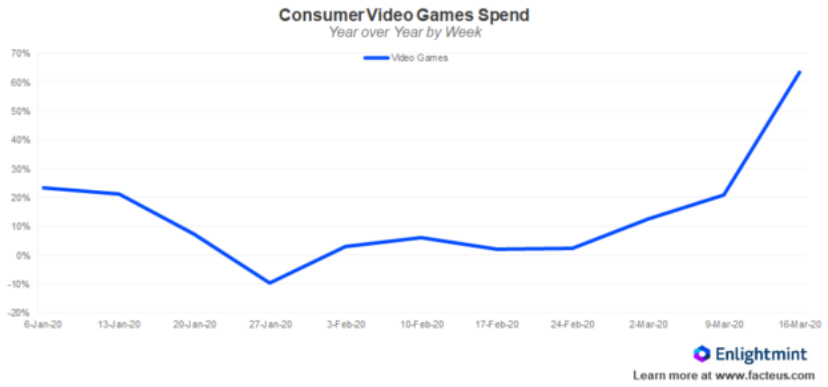
فهرست جداول

فهرست معادلات و فرمول‌ها

فصل ۱
مقدمه

مقدمه

امروزه با توجه به اوضاع همه‌گیری ویروس کووید-۱۹ [1]و همینطور زندگی مدرن و شهری، استرس زندگی بیشتر و بیشتر می‌شود [۱]. در این میان مردم سعی می‌کنند با دست و پا کردن یک سرگرمی برای خود از این استرس، هرچند برای زمان کمی دوری کنند [۲, ۳]. یکی از پرطرفدارترین سرگرمی‌های انتخابی بازی‌های ویدئویی است که به دلیل دسترسی‌پذیری زیاد، هر روز توسط صدها میلیون کاربر، تجربه می‌شود [۴].

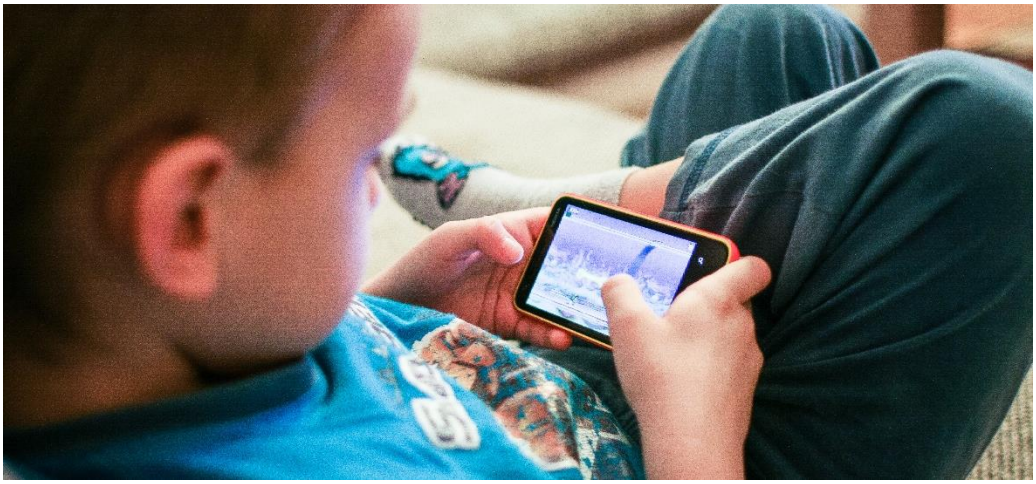


شکل ۱–۱ نمودار رشد خرید بازی‌های ویدئویی بعد از شروع قرنطینه خانگی [۲]

علاوه بر این همواره از بازی‌های ویدئویی به عنوان یک رسانه موثر یاد می‌شود [۵]. با گسترش صنعت بازی‌های ویدئویی، سرمایه‌گذاری ها در این زمنیه بیشتر شده [۶] و بازی‌ها علاوه بر سرگرمی، به دنبال هدف‌های بالاتری مانند آموزش از طریق روش‌های مختلف اجوتینمنت[2] و افزایش سطح آگاهی اجتماعی نیز هستند.

[1] Covid-19
[2] Edutainment

بازارهای نرم‌افزار تلفن‌های هوشمند، این امکان را فراهم آورده که توسعه‌دهندگان با سادگی بتوانند بازی‌های ساخته شده خود را در اختیار عموم قرار دهند و در این امر مشارکت داشته باشند. علاوه بر این توسعه‌دهندگان بازی برای اینکه به کارایی حداکثری دست پیدا کنند، همواره از ابزارهای آماده استفاده می‌کنند و استفاده از موتورهای بازی‌سازی آماده، همواره جزو پیشفرض‌های توسعه‌دهندگان بازی‌های رایانه‌ای بوده است.



شکل ۱–۲. در دسترس بودن بازی‌های ویدئویی در طول روز

موتورهای بازی‌سازی عمومی برخلاف موتورهای درون‌سازمانی[1] در اختیار عموم قرار داده می‌شوند و توسعه‌دهندگان را از پیاده‌سازی مجدد اجزای پرتکرار رها می‌کنند، اما به همین دلیل، نمی‌توانند روی مکانیزم‌های جزئی بازی‌های تمرکز کنند، از این رو نیاز به توسعه‌دهندگان ابزار که با مشارکت خود، موتورهای بازی‌سازی را غنی می‌سازد حس می‌شود.

علاوه بر این با توجه به بالا رفتن تعداد کاربران بازی‌های ویدئویی[۴] ، از صنعت انتظار می‌رود بازی-های جدید با مکانیزم‌های جدید را با سرعت نسبتا سریعی وارد بازار کنند. به همین دلیل همواره در صنعت بازی‌سازی بخش مهمی از بودجه به توسعه چارچوب‌های[2] قابل استفاده مجدد اختصاص می-یابد که مکانیزم‌های اساسی بازی‌ها، از ابتدا آماده استفاده باشد.

[1] In-house
[2] Framework

در این میان شرکت‌های سازنده موتورهای بازی‌سازی عمومی پراستفاده، برای کاربران خود، بستر-هایی[1] بصورت بازارچه ارائه می‌دهد تا توسعه‌دهندگان ابزار و هنرمندان بازی، محصولات خود را برای استفاده مجدد به فروش بگذارند یا بصورت رایگان در اختیار عموم قرار دهند [۷]. از این محصولات می‌توان به بسته‌های هنری از صدا و موسیقی اشاره کرد که به شدت قابل استفاده مجدد است. نوع دیگری از این محصولات کاراکترهای استخوان‌بندی شده[2] سه بعدی که کمبود هنرمندان در صنعت بازی‌سازی را جبران می‌کند. در نهایت محصولاتی که بصورت ابزار توسعه و چارچوب روی موتورهای بازی‌سازی ارائه می‌شود، با هدف کاهش زمان توسعه محصول در اختیار توسعه‌دهندگان قرار می-گیرد.

برش اشیای سه‌بعدی یکی از مکانیزم‌های پرطرفدار در بازی‌های ویدئویی است که علاوه بر بازی‌های موبایلی غیرجدی[3]، در بازی‌های جدی تریپل ای[4] نیز مورد استفاده قرار می‌گیرد. علاوه بر این محاسباتی که در برش انجام می‌شود قابل استفاده در نمایش مکان برش است که -با فرض اینکه در بازی می‌خواهیم یک کاراکتر را برش بدهیم- در بازی‌هایی که برای سنین پایین تر ارائه می‌شود، مورد استفاده است.

در دسته بازی‌های غیرجدی که از این مکانیزم پرطرفدار استفاده می‌کنند، می‌توان به بازی موبایلی Fruit Ninja[۸] و Sword Play[۹] اشاره کرد که مورد اول فقط در بازارچه گوگل، بیش از ۵۰۰ میلیون دانلود دارد [۱۰]. در دسته بازی‌های جدی می‌توان به بازی طلوع چرخ‌دنده آهنین: انتقام[5] اشاره کرد که در سال ۲۰۱۲ عرضه شد و مکانیزم برش جزو مکانیزم‌های هسته این بازی بود. تخمین شده این بازی در فروشگاه بازی‌های کامپیوتری استیم و شبکه پلی‌استیشن ۳ بیش از ۱.۲ میلیون کپی به فروش رسانده است [۱۱, ۱۲].

[1] Platforms
[2] Rigged
[3] Casual
[4] AAA
[5] Metal Gear Rising: Revengeance

شکل ۱–۳. صحنه‌ای از بازی Fruit Ninja

۱-۱-تعریف پروژه

در این پروژه قصد دارم یک چارچوب آماده قابل استفاده مجدد برای بریدن اشیای سه بعدی در موتور بازی‌سازی یونیتی[1] توسعه دهم. متاسفانه بدلیل مسائل سیاسی [۱۳]، این ابزار قابل انتشار در فروشگاه یونیتی قابل عرضه نیست؛ در عوض این چارچوب بصورت متن باز در اختیار عموم قرار داده می‌شود و بصورت رایگان قابل دانلود و استفاده خواهد بود. نمونه‌ای از استفاده این چارچوب در یک دمو بصورت خروجی قابل اجرای ویندوز به عنوان اثبات کار ارائه می‌شود.

برش شی سه‌بعدی در یک محیط گرافیکی، یکی از مسایل گرافیک کامپیوتری است که ورودی آن، توری سه‌بعدی m و صفحه برش p است و خروجی آن حداکثر دو توری سه‌بعدی است که نتیجه‌ی برش توری m توسط صفحه‌ی p است، بصورتی که توپولوژی[2] شی از ریخت نیفتاده باشد و سطح برش و سایر سطوح مانند انتظار تولید شده باشند.

در این چارچوب، توسعه دهنده می‌تواند با ساخت یک شی در ادیتور موتور بازی‌سازی، ویژگی قابل برش بودن را به آن اضافه کند و با فراخوانی متد برش با ورودی صفحه‌ی برش شی مورد نظر را برش دهد.

[1] Unity
[2] Topology

اشیای قابل برش، می‌توانند توپر یا توخالی باشند و همینطور پس از برش، اعمال فیزیک روی جسم بریده شده قابل تنظیم است. همچنین تنظیم صفحه‌ی برش می‌تواند با استفاده از توابع کاربردپذیر[1] با روش‌های مختلفی انجام شود.

۱-۲-چالش‌ها

اولین چالش این پروژه، بودجه زمانی آن است. برخلاف تصاویر تولیدشده توسط کامپیوتر، بازی‌ها بصورت بلادرنگ[2] اجرا می‌شوند و -حداقل در زمان حال- نمی‌توان به سیستم‌های ابری برای این‌کار تکیه کرد. به همین دلیل باید در نظر گرفت که سخت‌افزاری که این الگوریتم را اجرا می‌کند تا چه حد قادر به انجام این محاسبات است. بازی‌های ویدئویی باید بتوانند روی طیف گسترده‌ای از دستگاه ها قابل اجرا باشند تا رسالت خود را به پایان برسانند.

سرعت اجرای بازی روی دستگاه‌های مختلف با واحد فریم بر ثانیه[3] اندازه گیری می‌شود و واحد زمانی محاسبات در بازی‌ها همین فریم‌ها هستند. اگر یک دستگاه برای محاسبه‌ی یک فریم زمان زیادی ببرد، فریم بعدی با فاصله‌ی زمانی بیشتری روی صفحه نمایش نشان داده می‌شود و نرخ فریم کاهش می‌یابد و در اصطلاح بازی از کاربر عقب می‌افتد[4].

هر چه سخت‌افزار قوی‌تری داشته باشیم، می‌توانیم به نرخ فریم بالاتری دست پیدا کنیم؛ زیرا سخت-افزارهای قوی زمان کمتری برای انجام محاسبات صرف می‌کنند. معادله ۱-۱ محاسبه نرخ فریم را نشان می‌دهد.

$$framerate(frame\ per\ second) = \frac{1}{time\ to\ calculate\ one\ frame(s)}$$ معادله ۱-۱

[1] Utility functions
[2] Real-time
[3] Frame per second (FPS)
[4] Lag

نرخ فریم[1] قابل قبول برای بازی‌های موبایلی ۳۰، برای کنسول‌های خانگی و کامپیوترهای شخصی ۶۰، و برای بازی‌های واقعیت افزوده[2] (که سخت‌افزارشان با کمک کنسول‌های خانگی و کامپیوترهای شخصی قدرت گرفته است) ۹۰ است. این یعنی به ترتیب برای دستگاه‌های گفته شده، ۳۳، ۱۶ و ۱۱ میلی‌ثانیه زمان در اختیار داریم تا از نظر نرخ فریم تجربه کاربری خوبی به کاربر منتقل کنیم [۱۴].

علاوه بر این باید این نکته را در نظر گرفت که هر فریم محاسبات دیگری از جمله کد موتور بازی-سازی و محاسبات فیزیک (که ممکن است بخاطر کارایی بهتر، هر چند فریم انجام شود) نیز در هر فریم انجام می‌شود تا پس از آن گذر محاسبات انجام شده از واحد پردازنده مرکزی به کارت گرافیک برای رندر و نمایش روی صفحه نمایش انجام گیرد.

چالش دیگر حفظ توپولوژی شی مورد برش است؛ یعنی جسم مورد برش نباید از شکل افتاده باشد. برای اینکه این موضوع اثبات شود، با فرض اینکه پس از برش، دو توری خروجی در جای خود باقی بمانند، هیچ تفاوتی در تصویر رندر شده بعد و قبل از برش نباید وجود داشته باشد. این چالش شامل حفظ داده‌ی رئوس و یو وی و نرمال آن‌هاست که در فصل چهارم به این مفاهیم پرداخته می‌شود.

چالش بعدی توپر کردن شی ایجاد شده است که در گرافیک کامپیوتری به مسئله مثلثی‌سازی توری[3] معروف است. در این مسئله از نظر پیچیدگی زمانی مثلثی‌سازی گلوگاه برش اشیاء است و باید علاوه بر سرعت زیاد با دقت زیادی نیز انجام شود؛ چرا که دستکاری توری ظرافت‌هایی دارد و اشتباهات هرچند کوچک به راحتی در خروجی قابل نمایش خواهند بود. برای مثلثی‌سازی باید رئوس[4] جدید پردازش شوند تا بهترین شکل و با یک توپولوژی معقول رئوس جدید به شکل نهایی اضافه شود.

[1] Framerate
[2] Virtual Reality (VR)
[3] Mesh triangulation
[4] Vertex

مورد دیگر تشخیص برخورد صفحه برش با اشیاء است. این کار می‌تواند به راحتی توسط توسعه-دهندگان انجام شود، ولی برای ساخت یک چارچوب کامل و قابل استفاده مجدد، تمام ابزارهای مفید احتمالی باید در اختیار توسعه دهندگان قرار گیرد. تشخیص برخورد در محیط‌های سه‌بعدی با اجزایی به نام برخوردکننده‌ها انجام می‌گیرد که معمولا توسط موتورها در اختیار توسعه‌دهندگان قرار می‌گیرد اما برای کارایی بهتر نیاز به پیش‌پردازش‌هایی داریم تا بدون محاسبه برخورد با کمک موتور فیزیکی، اشیای مورد برخورد را تشخیص دهیم.

۱-۳-ساختار پایان‌نامه

در این بخش، به ساختار پایان‌نامه پرداخته می‌شود و فصل‌های بعدی به اختصار معرفی می‌شوند. در ابتدا ابزارهای مورد استفاده به تفصیل معرفی می‌شوند تا آشنایی کامل با آنها صورت گیرد و دلیل استفاده از این ابزارها آورده می‌شود تا انتخاب های صورت گرفته را توجیه کند. سپس کارهای صورت گرفته برای حل این مسئله معرفی و بررسی می‌شود و کمبودهای آنها مطرح می‌شود. در فصل چهارم برای زمینه‌ی بهتر مفاهیم گرافیک کامپیوتری که در این پروژه مورد نیاز است معرفی می‌شود تا خواننده ذهنیت کاملی از مسئله داشته باشد. پس از آن الگوریتم‌های مورد استفاده برای برش اشیاء و مثلثی سازی آن‌ها و همینطور سایر الگوریتم‌های و تکنولوژی‌های مورد استفاده برای این ابزار معرفی می‌شود. در فصل ششم جزئیات پیاده‌سازی مطرح می‌شود و نحوه حل چالش‌های مطرح شده بصورت کامل توضیح داده خواهد شد و در فصل هفتم زمان اجرای الگوریتم روی اشیای مختلف بررسی می‌شود. در نهایت به جمع‌بندی و نیجه‌گیری پرداخته می‌شود و پیشنهادات بهبود این ابزار مطرح می‌شود.

فصل ۲

معرفی و توجیه ابزارهای استفاده شده

معرفی و توجیه ابزارهای استفاده شده

در این فصل ابزارها، چارچوب‌ها و زبان‌های برنامه‌نویسی استفاده شده معرفی می‌شود و قابلیت‌های آنها مطرح میشود و در ادامه استفاده از آنها توجیه می‌شود.

در این پروژه از موتور بازی‌سازی یونیتی، زبان سی‌شارپ[1] و اچ‌ال‌اس‌ال[2] استفاده شده که در ادامه به تفصیل به آنها پرداخته می‌شود.

۲-۱-موتور بازی‌سازی یونیتی

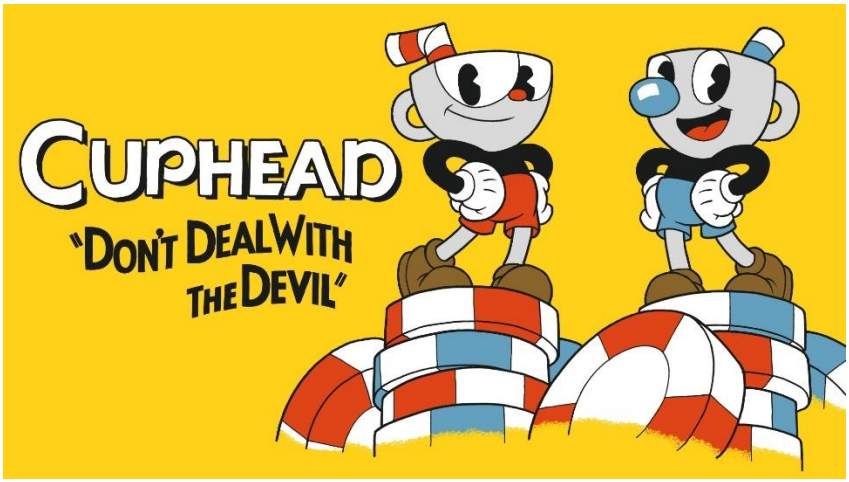
شکل ۲-۱ لوگوی موتور بازی‌سازی یونیتی

یونیتی یک موتوربازی سازی چند بستری است که قابلیت ساخت بازی روی سیستم عامل‌های ویندوز مایکروسافت، مک او اس و لینوکس را دارد. این موتور بازی‌سازی قادر به خروجی گرفتن روی بسترهای مختلفی از جمه اندروید و آی او اس، ویندوز و مک او اس و لینوکس، ایکس باکس و پلی استیشن و تی‌وی او اس می‌باشد [۱۵].

این موتور از واقعیت مجازی و واقعیت افزوده نیز پشتیبانی می‌کند که بدلیل کمبود پشتیبانی در این زمینه در موتورهای عمومی دیگر، یونیتی تبدیل به یکی از نامزدهای اصلی انتخاب برای توسعه-دهندگان واقعیت مجازی شده است. همه این‌ها باعث شده تعداد زیادی از توسعه‌دهندگان با هدف انتشار همزمان بازی خود روی بسترهای مختلف به این موتور روی بیاورند [۱۶].

[1] C#
[2] High Level Shader Language (HLSL)

این موتور بازی‌سازی می‌تواند بازی‌های دو و سه بعدی را بصورت محلی و بدون استفاده از ابزارهای دیگر بسازد. بازی‌های معروف زیادی روی این موتور بازی‌سازی ساخته شده است که می‌توان در این میان به ندای وظیفه: موبایل[1]، کله‌فنجونی[2] و فرقه قاتلین: اتحاد[3] اشاره کرد [۱۵].



شکل ۲–۲. تصویر تبلیغاتی بازی کله‌فنجونی

همه این‌ها باعث می‌شود که یونیتی در بین سایر موتورهای بازی‌سازی انتخاب اول برای استودیوهای کوچک باشد. ۵۰ درصد کل بازی‌های ساخته شده در موبایل، کامپیوترهای شخصی و کنسول‌های بازی با یونیتی ساخته می‌شود و ۷۱ درصد از بهترین ۱۰۰۰ بازی موبایلی، با این موتور ساخته شده است [۱۷]. همچنین ۴۷ درصد از توسعه دهندگان نرم‌افزار از یونیتی استفاده می‌کنند؛ پس برای ساخت یک ابزار توسعه که توسط توسعه‌دهندگان زیادی مورد استفاده قرار گیرد، این موتور بهترین انتخاب است [۱۸].

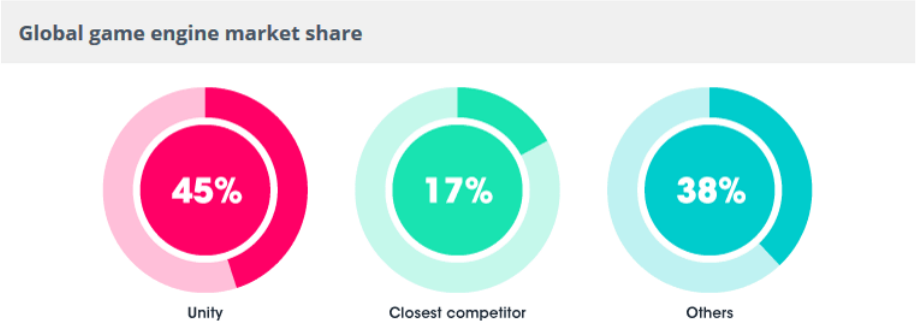
۲-۱-۱-مزایای موتور بازی‌سازی یونیتی

همانطور که اشاره شد، موتور بازی‌سازی یونیتی از بسترهای زیادی پشتیبانی می‌کند، این باعث می‌شود که بتوان از یک کد بیس را در بسترهای مختلف بدون نگرانی اضافی در مورد چگونگی مهاجرت به بسترهای دیگر، خروجی گرفت.

[1] Call of Duty: Mobile
[2] Cuphead
[3] Assassin's Creed: Unity

یونیتی جامعه توسعه‌دهندگان بزرگی دارد. برای همین آموزش‌های بسیار زیادی برای تازه‌کاران در زمینه بازی‌سازی موجود است که می‌توانند چراغ راه آنها در شروع مسیر بازی‌سازی باشد. علاوه بر این وبسایت‌های زیادی[1] در زمینه پاسخ به نیاز توسعه‌دهندگان توسط شرکت یونیتی گیمز ساخته شده که استفاده حداکثری از این جامعه بزرگ صورت گیرد.

درصد زیادی از توسعه‌دهندگان بازی جوان، برای شروع به این موتور روی می‌آورند و تا وقتی یونیتی پاسخ نیازهای آنها را بدهند به آن می‌چسبند و همین باعث می‌شود برای کاهش هزینه مهاجرت به یک موتور دیگر ابزارهای متفاوت زیادی روی این موتور توسعه داده شود و این موتور روز به روز قدرتمندتر می‌شود.



شکل ۲–۳ سهم بازار کاربران برای موتورهای بازی‌سازی [۱۸]

یونیتی با ارائه‌ی ابزار آنالیز، اجازه می‌دهد که تحلیلگران داده بتوان از کاربران درون‌بینی داشته باشند و با تحلیل داده‌های بدست آمده برای بهبود بازی پیشنهادهایی ارائه دهند که با توجه به آن، توسعه-دهندگان، فرمول‌های سند طراحی بازی را تغییر دهند و یا تایید کنند [۱۹].

یونیتی برای موتور فیزیکی از موتور هَوِک[2] استفاده می‌کند که موتور بسیار قدرتمندی است و با کارایی مناسب ابزارهای زیادی در اختیار کاربران قرار می‌دهد. این این ابزارهای می‌توان به پرتو افکن[3] و برخوردکننده‌ها[4] اشاره کرد.

[1] answers.unity.com, forum.unity.com
[2] Havok
[3] Raycaster
[4] Collider

یونیتی همچنین از معماری ECS[1] که باعث کارایی بهتر در زمان اجرا می‌شود استفاده می‌کند که قدرت زیادی به آن در بازی‌های موبایلی می‌دهد.

یونیتی با استفاده از سیستم شغل[2]، قادر به استفاده از ریسمان‌های دیگر واحد پردازنده می‌باشد که در استفاده حداکثری از منابع و افزایش کارایی کمک شایانی می‌کند.

۲-۱-۲-معایب موتور بازی‌سازی یونیتی

کمبود قالب آماده شروع پروژه برای یونیتی باعث شده، توسعه‌دهندگان -مگر اینکه قالب درون سازمانی داشته باشند- همواره باید پروژه جدید را از صفر شروع کنند.

همه‌منظوره بودن این موتور باعث شده، بسیاری از ابزارها که در خیلی از بازی‌ها باید استفاده شود، نیمه پخته باشد و عملا در مقیاس بزرگ قابل استفاده نباشد.

۲-۲-زبان سی‌شارپ

بدلیل اینکه بازی‌های ویدئویی منابع زیادی نیاز دارند، مورد استفاده ترین زبان‌ها برای این‌کار زبان-های نسبتا سطح پایین تر مثل سی پلاس پلاس[3] است. یونیتی با فرض اینکه توسعه‌دهندگان تازه‌کار را هدف گرفته، تصمیم گرفته بجای سی پلاس پلاس که یک زبان مدیریت نشده[4] است و جمع‌آوری زباله[5] خودکار ندارد، از سی شارپ بصورت رسمی پشتیبانی کند. البته با استفاده از سی پلاس پلاس و کتابخانه‌های مرتبط پویا[6] هم می‌توان در یونیتی برنامه‌نویسی کرد ولی با این کار استفاده از اجزای درونی یونیتی سخت می‌شود.

یونیتی برای اینکه بتواند روی بسترهای مختلف خروجی یکسان داشته باشد، از مونو دات نت[7] استفاده می‌کند که یک پیاده‌سازی متن باز از چارچوب دات نت است که قابلیت خروجی روی بستر-های مختلف را دارد. به همین دلیل توسعه‌دهندگان یونیتی باید از سی‌شارپ استفاده کنند و انتخاب‌های دیگر در این زمان از کارایی لازم برخوردار نیست.

[1] Entity Component System
[2] Job System
[3] C++
[4] Unmanaged
[5] Garbage collection
[6] Dynamic link library
[7] Mono .Net

استفاده از مونو، گرچه قدیمی است، باعث می‌شود توسعه دهندگان به فکر کد بیس‌های متفاوت برای بسترهای متفاوت نباشند.

۲-۲-۱-مزایای سی‌شارپ

زبان سی شارپ یک زبان شی‌گرا است که به دلیل مستندات قوی، می‌تواند انتخاب خوبی برای تازه-کاران باشد و به دلیل شباهت نحوی به زبان‌های پرطرفداری مثل جاوا، کسانی که تجربه برنامه-نویسی دارند در زمان کمی با این زبان به آشنایی نسبی دست پیدا می‌کنند که برای توسعه بازی کافی است.

از دات نت چهار زبان سی‌شارپ می‌تواند با اجرای همگام[1] از گیر کردن نرم‌افزار جلوگیری کند [۲۰].

۲-۲-۲-معایب سی‌شارپ

سی‌شارپ به دلیل اینکه دسترسی زیادی به سطح سخت‌افزاری ندارد، روی کارایی سیستم کنترل کمی دارد.

نسبت به زبانی مثل سی پلاس پلاس، سی شارپ فقط برای کارایی خوب روی سیستم عامل ویندوز طراحی شده و در کل در مواردی مثل بازی که یک میلی‌ثانیه ارزشمند است سربار محاسباتی زیادی دارد.

سی‌شارپ کتابخانه‌هایی برای کار راحت با آرایه‌ها ارائه می‌دهد و یکی از معروف ترین آن‌ها LinQ است که از نظر کارایی، بسیار کند است [۲۱].

۲-۳-زبان اچ ال اس ال

زبان اچ ال اس ال یک زبان سایه‌زنی سطح بالا است که توسط ماکروسافت برای دایرکت اکس[2] توسعه داده شده است که مشابه زبان جی ال اس ال[3] برای اوپن جی ال[4] است [۲۲].

زبان سایه‌زنی یک زبان برنامه‌نویسی گرافیکی است که برای نوشتن اثرهای بصری در محیط‌های گرافیکی استفاده می‌شود [۲۲].

[1] Async
[2] DirectX
[3] OpenGL Shading Language (GLSL)
[4] OpenGL

یونیتی با استفاده از شیدر لَب[1]، اجازه‌ی استفاده از زبان‌های مختلف سایه‌زنی را می‌دهد که برای سخت‌افزارهای گرافیکی مختلف مورد استفاده قرار می‌گیرد. از این زبان‌ها می‌توان به جی ال اس ال و سی جی[2] اشاره کرد. لازم به ذکر است استفاده از این زبان صرفا برای نمایش مکان برخورد شی با صفحه بوده است و به پیاده‌سازی برش مربوط نیست. در این مورد در فصل شش توضیحات بیشتری ارائه خواهد شد.

دلیل اصلی استفاده از این زبان، ترجیح شخصی بوده، ولی به طور کلی زبان‌های سایه‌زنی مدرن به راحتی به هم تبدیل می‌شوند و بعضا یکسان هستند ولی فقط برای بسترهای هدف متفاوتی توسعه داده شده اند. این پروژه برای دمو خروجی ویندوز دارد و یکی دیگر از دلایل استفاده از این زبان، همین سازگاری است.

[1] Shader Lab
[2] CG

فصل ۳
مروری بر کارهای گذشته

مروری بر کارهای گذشته

در این فصل به مرور کارهای گذشته برای حل این مسئله پرداخته می‌شود. با توجه به اینکه هدف از این پروژه توسعه یه ابزار تولید برش با هدف استفاده از موتورهای بازی‌سازی می‌باشد، تحقیقات تئوری و ابزارهای توسعه داده شده که خارج از موتورهای بازی‌سازی و یا در موتورهای بازی‌سازی درون سازمانی که در اختیار عموم نیست، استفاده می‌شوند، معرفی نمی‌شود.

۳-۱-افزونه ایزی اسلایس[1]

ایزی اسلایس یک ابزار متن باز توسعه داده شده با یونیتی است که به صورت رایگان در اختیار عموم قرار داده شده است [۲۳]. این ابزار با هدف کارایی بالا توسعه داده شده است و با ورودی صفحه و شی مورد برش عملیات برش را روی اشیای محدب ساده انجام می‌دهد، در فصل چهار درباره انواع اشیای سه بعدی و توری‌های محدب و مقعر[2] توضیح خواهد داده شد.

یکی از مشکلات این ابزار این است که حتما شی مورد برش را توپر می‌کند و به توسعه‌دهنده انتخابی نمی‌دهد. با این وجود بخاطر کد تمیز و قابل خوانا، می‌توان این ویژگی را به این چارچوب اضافه کرد، ولی همانطور که در فصل پنج خواهید خواند، تولید شی توخالی در موتور یونیتی، پیچیدگی‌های خاص خودش دارد ولی عملا توسعه یک سیستم تولید شی توخالی زمان کمتری از ساختن روی این چارچوب را می‌برد.

علاوه بر این ابزارهای تشخیص برخورد در خورد چارچوب قرار داده نشده است که باعث می‌شود توسعه دهنده زمان اضافی را برای توسعه سیستم تشخیص شی مورد برش صرف کند. این ابزار به این گونه عمل می‌کند که شی مورد برش و صفحه بُرنده باید در اختیار مولفه‌ی برنده قرار گیرد ولی هیچ تابع کاربردپذیری برای محاسبه‌ی شی و صفحه در این چارچوب قرار داده نشده است.

[1] EzySlice
[2] Concave

نکته‌ی مورد تامل دیگر این است که این ابزار در سال ۲۰۱۵ توسعه داده شده است و به طور رسمی فقط تا یونیتی ۲۰۱۸ پشتیبانی می‌شود. با وجود اینکه نسخه های جدید یونیتی تفاوت زیادی ندارند ولی با توجه به آپدیت‌های مرتب و تغییر API های موتور بازی‌سازی امکان اینکه این ابزار کارایی روز قبل خود را نداشته باشد، کم نیست و با پشتیبانی ضعیف، ریسک استفاده از ابزارهای میراث[1] زیاد می‌شود.

۳-۲-گره برش توری رویه ای آنریل[2]

این ابزار بصورت بومی در موتور بازی‌سازی آنریِل موجود است و توانایی برش اشیای محدب ساده را دارد. همچنین این ابزار قابلیت اضافه کردن "درپوش" روی اشیای تولید شده را دارد که باعث می‌شود جسم برش زده شده، توپر باشد [۲۴].

این ابزار فقط برای موتور بازی‌سازی آنریِل توسعه داده شده و در موتورهای دیگر قابل استفاده نیست. همانطور که در فصل دوم گفته شد، موتور یونیتی جامعه‌ی بسیار بزرگتری دارد و این ابزار مفید فقط با داده توری بومی موتور بازی‌سازی آنریِل کار می‌کند و برای استفاده در دیگر چارچوب‌ها نیازمند تغییرات فراوان است.

۳-۳-افزونه مش اسلایسر[3]

این ابزار که بهترین ابزار در این لیست است در فروشگاه یونیتی با قیمت ۷۰ دلار که برای یک ابزار تک منظوره قیمت بسیار زیادی است در دسترس است. این ابزار توانایی برش اشیای مقعر ساده بصورت همگام را دارد ولی از نظر زمانی پیچیدگی بیشتری دارد [۲۵].

ابزار مش اسلایسر بصورت مرتب آپدیت می‌شود و مشکلات فراوانی را بعد از ۵ سال برطرف می‌کند که نشان از پایه ریزی بد این پروژه دارد. همچنین استفاده از این ابزار به سرعت دیگر ابزارها نیست و به دلیل پیچیدگی زیاد، استفاده از این ابزار و آموزش آن زمان زیادی می‌برد [۲۵].

علاوه بر این، این ابزار با این فرض طراحی شده است که دستگاه مورد استفاده از این ابزار از قدرت سخت‌افزاری نسبتا خوبی برخوردار است.

[1] Legacy
[2] Unreal Procedural Mesh Slice
[3] Mesh Slicer

این ابزار به تازگی از اشیای استخوان‌بندی شده نیز پشتیبانی می‌کند که باعث وابستگی زیادی به دیگر ابزارها شده است و به همین دلیل برای انتشار در بازار قابل اطمینان نیست.

نکته‌ی دیگر این است که مستندات این ابزار بسیار ضعیف است و کد بسیار ناخوانایی دارد و بخاطر آزمون و خطای فراوان، بسیاری از کدهای نوشته شده استفاده نمی‌شود.

فصل ۴

معرفی اصول رایانش سه‌بعدی در بازی‌های ویدئویی

معرفی اصول رایانش سه بعدی در بازی‌های ویدئویی

در این فصل به معرفی پایه‌ای رایانش سه‌بعدی در گرافیک کامپیوتری و بازی‌های ویدئویی می‌پردازیم. در ابتدا اجزای سازنده‌ی یک شی را در یک محیط سه‌بعدی معرفی کرده و با کمک این با کمک داده توری یک شی، آن را در محیط سه‌بعدی بازنمایی می‌کنیم.

پس از آن درباره انواع توپولوژی‌های نمایش توری در گرافیک کامپیوتری توضیح خلاصه‌ای ارائه خواهد شد. سپس به تفاوت‌های یک شی سه‌بعدی در محیط سه‌بعدی بازی با محیط‌های گرافیکی دیگر پرداخته می‌شود و معرفی انواع مختلف اشیاء و سطوح سه‌بعدی صورت خواهد گرفت.

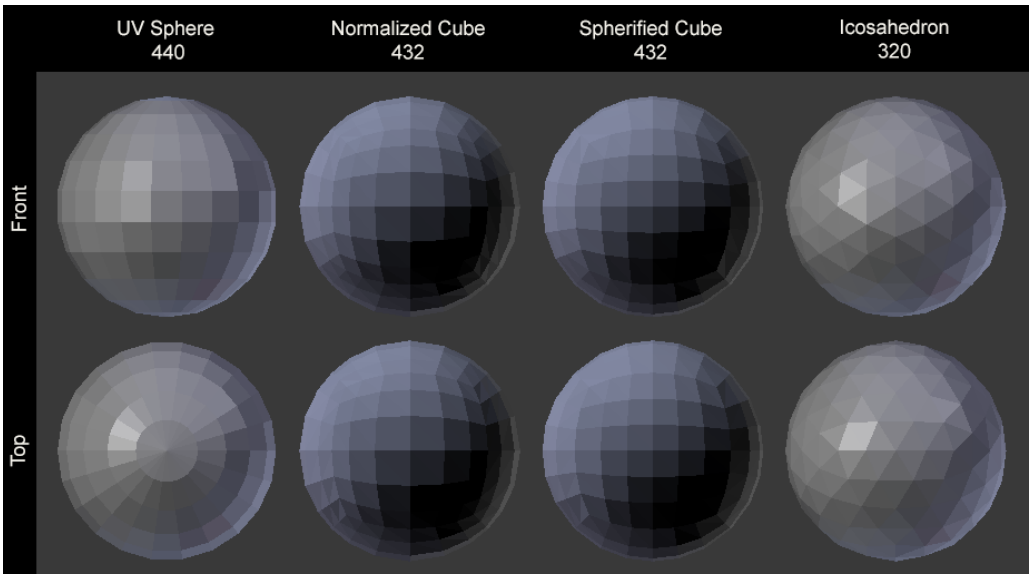
۴-۱-تعریف شی و توری

در یک محیط سه‌بعدی هر شی، شامل حداقل یک توری و یک جنس[1] است. هر شی می‌تواند به ازای هر توری (که در بازی‌سازی به آن زیرتوری[2] می‌گویند) یک جنس داشته باشد.

شکل ۴-۱ مقایسه محیط سه‌بعدی با و بدون نور و جنس

[1] Material
[2] Submesh

داده اصلی ذخیره شده در توری، تعیین کننده شکل یک شی است؛ برای مثال توری اولیه مکعب یا توری اولیه کره. لازم به ذکر است هر شکل می‌تواند توری‌های متفاوتی داشته باشد، برای مثال یک کره می‌تواند بصورت کره یو وی[1]، کره نرمال شده، ایکوساهدرون[2] و موارد دیگری نمایش داده شود. (شکل ۴-۲)

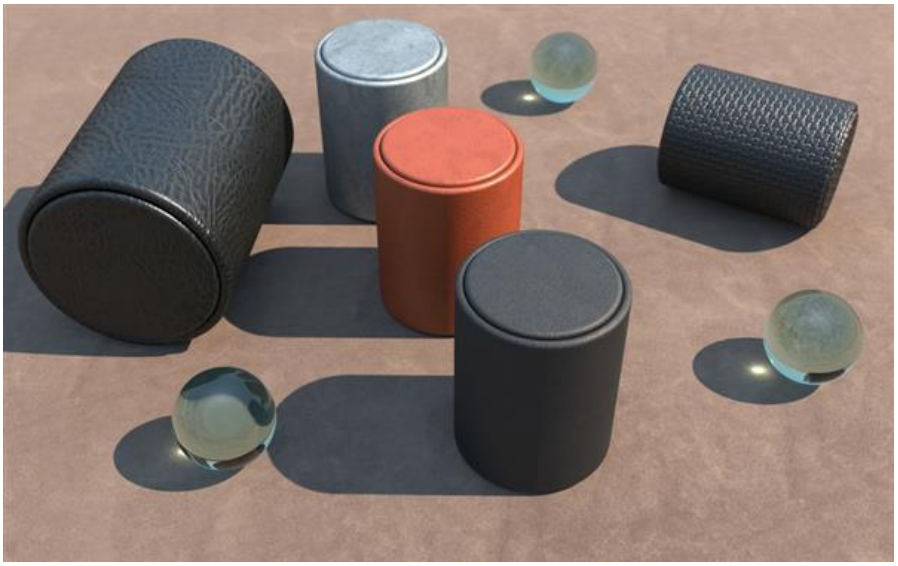


شکل ۴-۲ توپولوژی‌های مختلف ساخت یک کره در سه‌بعد [۲۶]

یک جنس تعیین کننده ویژگی‌های رندر شدن یک شی است. برای مثال یک کره می‌تواند قرمز یا آبی باشد و علاوه بر این نرم یا سخت باشد. ویژگی‌های یک جنس با توجه به سایه‌زن آن تعریف می‌شود، یک جنس می‌تواند ویژگی داشته باشد و یا اینکه بدون ویژگی باشد.

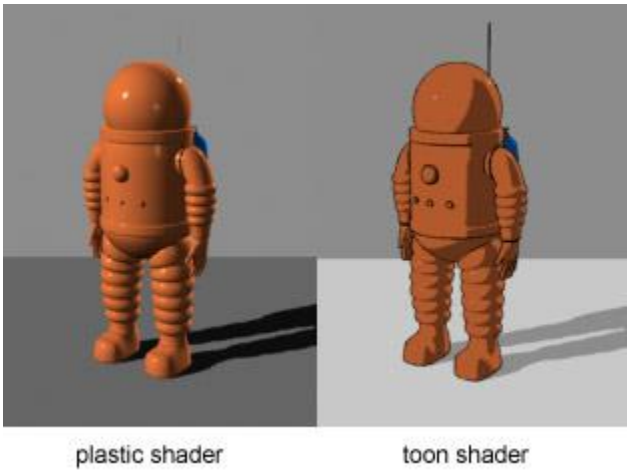
سایه‌زن‌های زیادی در محیط‌های سه‌بعدی استفاده می‌شود که می‌توان به سایه‌زن استاندارد که دارای ویژگی رنگ، آهنی بودن، زبر بودن، بافت[3]، بافت نرمال است اشاره کرد.

[1] UV Sphere
[2] Icosahedron
[3] Texture



شکل ۴–۳ جنس‌های مختلف روی اجسام سه‌بعدی گرافیکی

سایه‌زن معروف دیگری که در بسیاری از بازی‌ها مورد استفاده قرار می‌گیرد، سایه‌زن سلولی است که سایه‌ی پله‌ای دارد. در این سایه‌زن ها، ویژگی‌های از قبیل رنگ، رنگ سایه، تعداد پله‌های سایه دیده می‌شود. (شکل ۴-۴)



شکل ۴–۴ سایه‌زن پلاستیکی (چپ) در مقابل سایه‌زن کارتونی

۴-۲-اجزای سازنده توری سه‌بعدی

هر توری سه‌بعدی شامل رئوس است که هر راس توسط بردارهای مکان، یو وی، عمود و بیشتر بازنمایی می‌شود که در ادامه به معرفی آن‌ها میپردازیم. به طور کلی یک توری شامل لیستی از بردارهای دو و سه‌بعدی است و هر آرایه از این لیست‌ها به هم مربوط می‌شوند. برای مثال عضو سوم آرایه بردار مکان، ویژگی مکانی راس سوم را نمایش می‌دهد و عضو سوم آرایه بردار یو وی، موقعیت آن راس در تصویر بافت را نمایش می‌دهد.

۴-۲-۱-بردار مکان

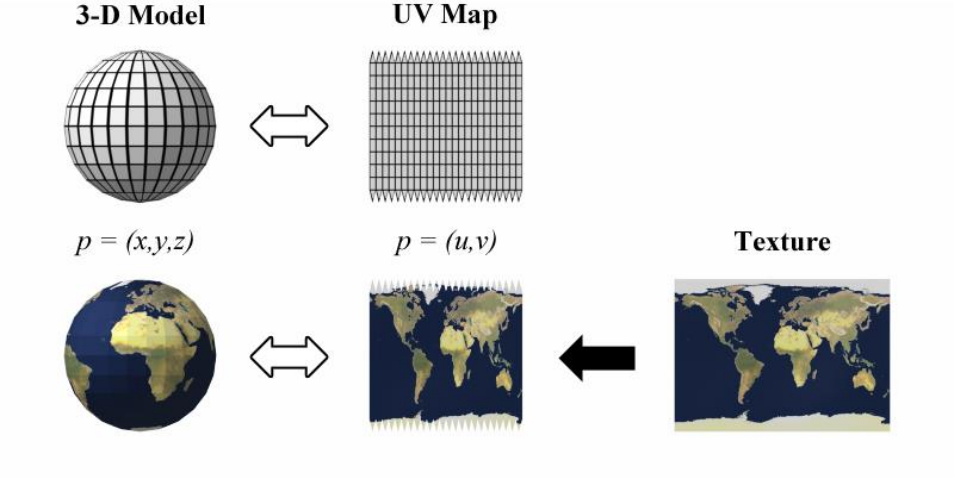
در سه‌بعد یک جسم توسط رئوس ساخته می‌شود که مکان رئوس با بردارهای سه‌بعدی نمایش داده می‌شود که تعیین کننده مکان آن راس هستند.

برای مثال یک چهارضلعی، چهار راس دارد که مکان این چهار راس شکل این چهارضلعی را تعیین می‌کند؛ این چهارضلعی می‌تواند با توجه به این مکان‌ها شکل‌های مختلفی به خود بگیرد.

۴-۲-۲-بردار یو وی

همانطور که گفته شد، هر شی می‌تواند یک یا چند جنس داشته باشد و هر جنس می‌تواند چندین بافت داشته باشد که برای موارد مختلف به کار می‌رود. برای اینکه بافت‌ها به درستی کار کنند، هر راس باید مکانی در تصویر بافت را نمایش بدهد که نیازمند واپیچی[1] رئوس روی یک تصویر را دارد. بردار یو وی، که برگرفته از حروف انگلیسی u و v است، تصویر بافت را در مختصات نرمال در دو بعد u و v نگاشت می‌کند و یو وی هر راس، نشان دهنده مکان آن راس در تصویر بافت است. (شکل ۴-۵)

[1] Unwrap



شکل ۴–۵ نگاشت مدل سه‌بعدی روی تصویر یو وی دوبعدی

۴-۲-۳-بردار عمود[1]

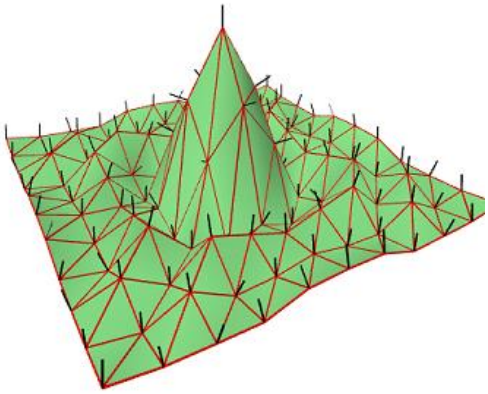
بردار عمود برای محاسبه‌ی نور پردازی و سایه‌زنی استفاده می‌شود. در یک سایه‌زن استاندارد میزان روشنایی یک راس با توجه به زاویه‌ی بین اشعه‌ی منبع نور و بردار عمود تعیین می‌شود. هرچه این زاویه کمتر باشد، نور بیشتری به این راس تابیده می‌شود.

در معادله ۴-۱ محاسبه شدت روشنایی راس v نشان داده شده است.

$$light\ of\ vertex\ v = (source\ light\ direction\ .\ v_{normal}) * source\ light\ intensity$$ معادله ۴–۱

در شکل ۴-۶ بردارهای عمود بصورت خط سیاه‌رنگ نمایش داده شده‌اند.

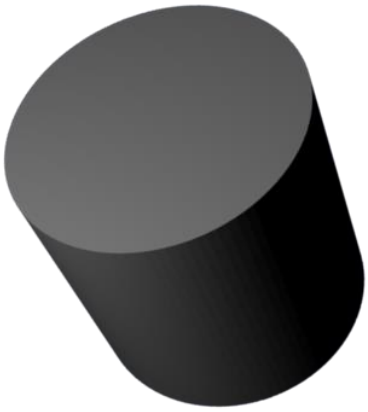
[1] Normal



شکل ۴–۶ بردار های عمود در یک نمودار سه‌بعدی

در نهایت روشنایی راس v در مقدار رنگ آن راس که توسط ویژگی‌های جنس آن شی تعیین می‌شود (از جمله رنگ و بافت) ضرب می‌شود تا برای رندر به کارت گرافیک گذر کند.

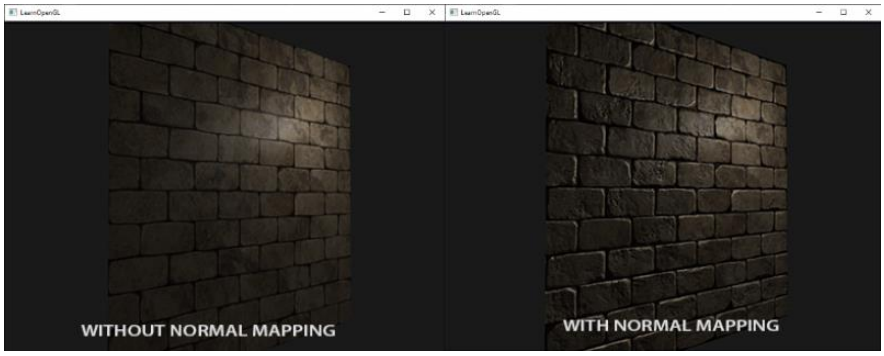
یک اتفاق ناگزیر این است که همواره یک راس ممکن است چند بردار عمود داشته باشد. این مشکل به این شکل حل می‌شود که آن راس در بردار رئوس چند بار تکرار می‌شود که هر بار در یک مثلث متفاوت قرار می‌گیرد ولی داده بردار عمود آن متفاوت است. با این تکنیک می‌توان لبه‌های تیز را در اشیاء بدون معرفی نورپردازی غیرواقعی داشته باشیم. (شکل ۴-۷)



شکل ۴–۷ لبه‌های تیز استوانه با رئوس دارای چند بردار عمود

۴-۲-۴-بردارهای دیگر

برای دستیابی به بافتی واقعی تر از شی، در گرافیک کامپیوتری از بافت عمودی[1] استفاده می‌کنند تا روی نورپردازی تاثیر داشته باشند(شکل ۴-۸). برای این کار هر راس به این تصویر نگاشت می‌شود و به یک بردار دو بعدی دیگر مانند یو وی نیاز داریم (که برای فشرده سازی معمولا از همان یو وی استفاده می‌شود) و علاوه بر آن برای هر راس به بردار مماس[2] و عمود مماس[3] استفاده می‌شود که از این بحث خارج است. برای مطالعه بیشتر سایت آموزشی اوپن جی ال منبع مناسبی برای شروع است.



شکل ۴-۸ تاثیر استفاده از تصویر عمود روی نورپردازی [۲۷]

در نهایت یک آرایه رئوس مختلف را به هم ربط می‌دهد. برای مثال در توپولوژی مثلثی، هر عوض این آرایه سه اندیس را نگهداری می‌کند که بیان‌کننده این است که کدام سه راس تشکیل یک مثلث می‌دهند. در نهایت با کمک این داده‌ها یک شی به کمک جنس خود قابل رویت در محیط سه‌بعدی می‌باشد.

۴-۲-۵-مکان، چرخش و اندازه

هر شی در سه بعد دارای سه ویژگی مکانی دیگر است که مستقل از داده‌ی توری آن جسم عمل می‌کند. در یک محیط سه‌بعدی دو جسم ممکن است داده توری یکسانی داشته باشند ولی با موقعیت، چرخش و اندازه متفاوت می‌توانند به اشیای یکتایی تبدیل شوند که در محیط سه‌بعدی قابل نمایش باشد.

[1] Normal Texture
[2] Tangent
[3] Bitangent

لازم به ذکر است با تغییر هر یک این سه ویژگی، هیچکدام از مولفه‌های توری تغییر نمی‌کند و مولفه‌های توری بر اساس موقعیتشان نسبت به نقطه مبدا (صفر مختصات) تعیین می‌شوند.

مکان، چرخش و اندازه هر کدام یک بردار سه‌بعدی هستند که در محیط سه‌بعدی ویژگی‌های مطلق آن شی را تعیین می‌کنند. در موتور بازی‌سازی یونیتی برای چرخش از مختصات چهارگانه[1] استفاده می‌شود که با یک بردار چهاربعدی نمایش داده می‌شود که به راحتی تبدیل به بردار سه‌بعدی می‌شود.

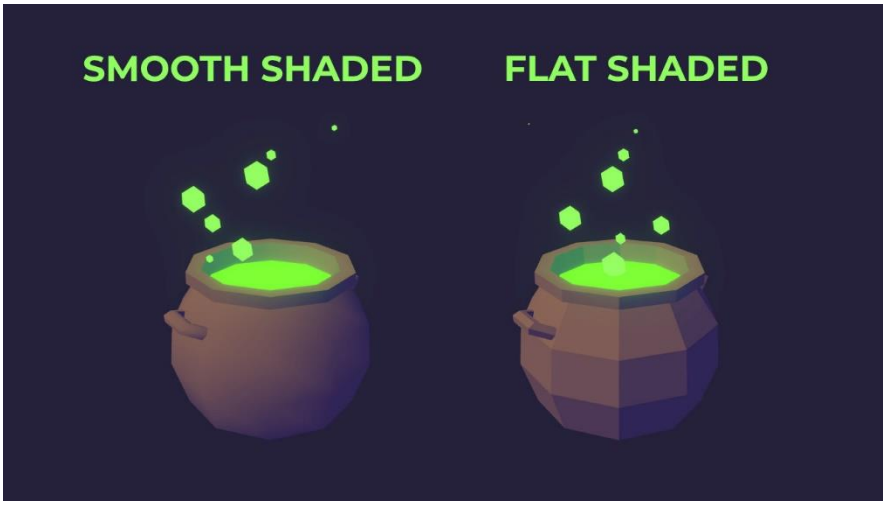
۴-۳-بازنمایی یک شی با کمک اجزای سازنده

برای نمایش یک شی تشکیل شده از رئوس بر روی صفحه معمولا نیاز به پردازش‌هایی روی داده توری داریم. این پردازش‌ها بسته به نیاز انجام می‌شوند و قبل از زمان اجرا انجام می‌شوند که سربار آن را روی خروجی نرم‌افزارهای بلادرنگ به صفر می‌رساند. در این بخش به معرفی دو مورد پرکاربرد می‌پردازیم.

۴-۳-۱-هموار سازی[2]

در این روش با الحاق خطی[3] بردارهای عمود به طوری تغییر می‌کنند با همان تعداد رئوس، شی بصورت نرم سایه بخورد و سایه پله‌ای نداشته باشد. این عملیات می‌تواند با افزایش رئوس نیز انجام شود که هزینه‌ی زیادی برای سیستم‌های بلادرنگ دارد ولی با این شگرد معروف، بدون نیاز به معرفی سطوح جدید می‌توان واقعی بودن سایه‌زنی را به مخاطب القا کرد. (شکل ۴-۹)

[1] Quarterion
[2] Smooth shading
[3] Linear interpolation



شکل ۴–۹ تاثیر هموار سازی بردار عمود در نورپردازی [۲۸]

۴-۳-۲-ساده سازی

در نرم‌افزارهای بلادرنگ یک شگرد دیگر برای کاهش استفاده از منابع، ساده‌سازی داده‌ی توری است. در بازی‌های ویدئویی این عملیات به این شکل است که قبل از خروجی گرفتن، از هر توری، چند سطح جزئیات[1] مختلف پخته می‌شود که در خروجی نهایی با توجه به فاصله دوربین از شی که از این توری استفاده می‌کند، توری‌های متفاوت جایگزین می‌شوند. هر چه دوربین از شی دورتر باشد، توری با جزئیات کمتر جایگزین می‌شود و هر چه دوربین به آن نزدیک شود، سطح جزئیات آن به سطح جزئیات توری اصلی میل می‌کند.

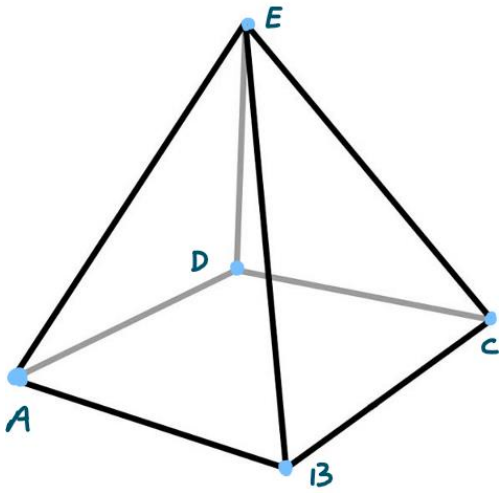
این ساده‌سازی با توجه به کاربرد می‌تواند در زمان اجرا باشد که به آن موزاییک‌کاری[2] گفته می‌شود [۲۹] که پویا تر از روش قبل است ولی منابع بیشتری می‌طلبد.

۴-۳-۳-مثال نمایش یک مکعب در محیط سه بعدی

قبل از پیشروی، در این بخش با توجه به مواردی که در بخش ۴-۲ گفته شد، مثالی از نمایش یک هرم در سه‌بعد را مرور می‌کنیم تا مطمئن شویم اصول نمایش سه‌بعدی در فضای گرافیکی به خوبی فهمیده شده باشد.

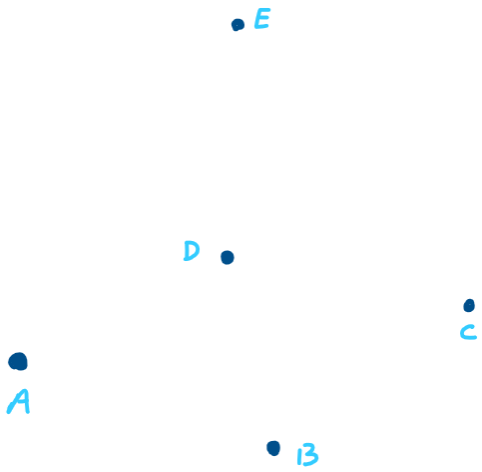
مطابق با شکل ۴-۱۰ یک هرم با یک نگاه ساده ۵ راس دارد.

[1] Level of details (LOD)
[2] Tessellation



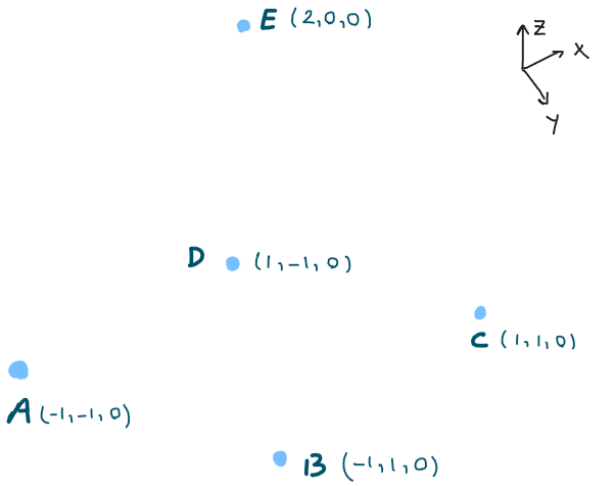
شکل ۴–۱۰ رئوس و اضلاع یک هرم سه‌بعدی

حال که با شکل کلی هرم آشنا شدیم، از صفر شروع می‌کنیم و ۵ نقطه A، B، C، D، E را میسازیم. ولی همانطور که در شکل ۴-۷ دیدیم، در شبیه‌سازی سه‌بعدی راس‌های یک چندضلعی برای نورپردازی و جنس واقعی‌تر، باید چند بار تکرار شوند.



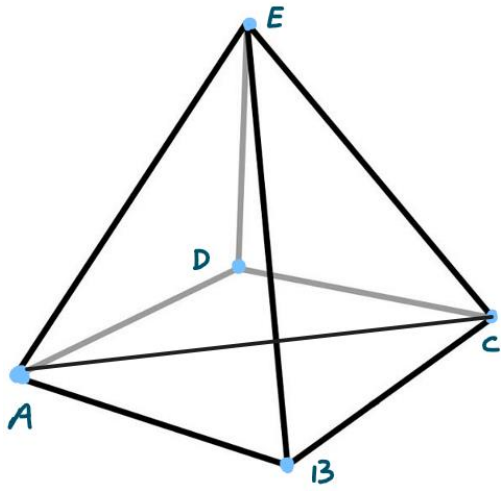
شکل ۴–۱۱ نقاط سازنده هرم در سه بعد

این پنج نقطه باید داده مکانی داشته باشند، برای همین باید موقعیت آن‌ها را داشته باشیم. برای راهنمایی در مورد جهات می‌توانید در شکل ۴-۱۲ از گوشه‌ی سمت راست بالا کمک بگیرید.



شکل ۴-۱۲ مکان نقاط سازنده هرم

برای اینکه تعداد دقیق رئوس را بدست آوریم می‌توانیم هرم شکل ۴-۱۳ را به مثلث‌هایی تجزیه کنیم.



شکل ۴-۱۳ مثلث های سازنده هرم

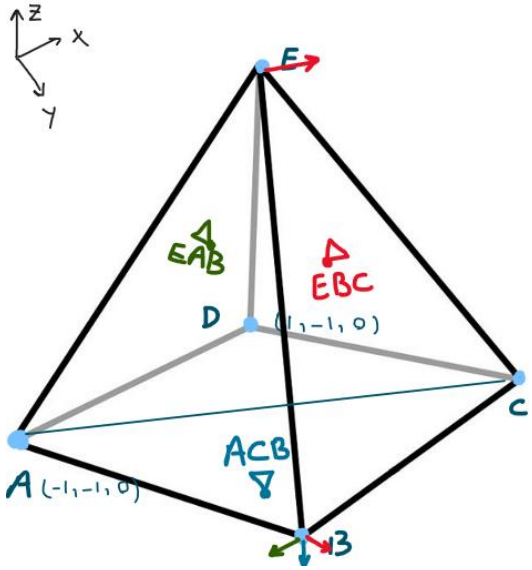
در شکل ۴-۱۳ دیده می‌شود که می‌توان یک هرم را با ۶ مثلث ساخت، به این شکل که چهار مثلث وجه‌های هرم را نشان می‌دهد و قاعده هرم از دو مثلث ساخته می‌شود. بنابراین در واقع نیاز به ۱۸ راس داریم. برای مثال در شکل ۴-۱۳ راس E چهار بار، در مثلث‌های EBC، ECD، EDA و EAB تکرار می‌شود. البته موقعیت مکانی راس E در هر چهار تکرار یکسان است.

در این مثال قرارداد می‌کنیم سمت جلو مثلث، سمتی است که رئوس آن بصورت پادساعتگرد مرتب شده‌اند. برای مثال اگر در آرایه اندیس‌ها مثلث شامل اضلاع A، B و C را بصورت ABC مرتب کنیم، زیر مثلث خالی می‌شود؛ زیرا از زیر بصورت ساعتگرد دیده می‌شود و پشت مثلث تشخیص داده می‌شود. نکته‌ی دیگر این است که ترتیب اضلاع اهمیتی ندارند و فقط جهت آنها برای موتور رندر حائز اهمیت است.

شش مثلث هرم ساخته شده برای نمایش درست باید به شکل زیر باشند.

- ACB
- ADC
- EBC
- ECD
- EDA
- EAB

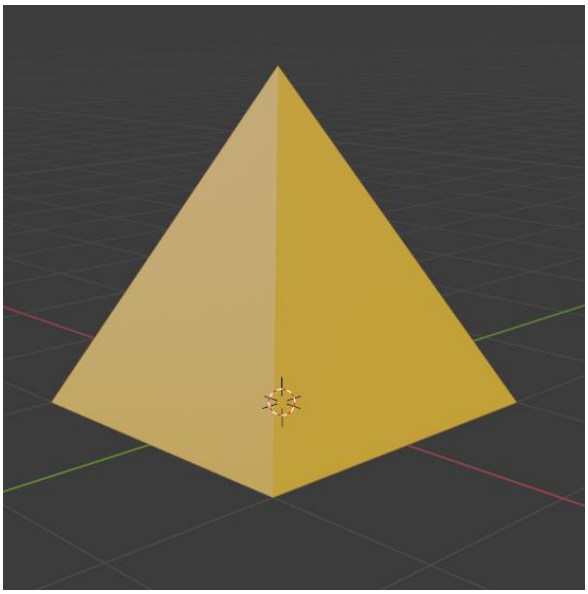
حال برای نورپردازی می‌بایست بردارهای عمود را اضافه کنیم. برای مثلا در شکل ۴-۱۴ راس B در سه مثلث مشترک است، پس نیاز به سه بردار عمود دارد. برای مثلث EBC (قرمز در شکل) چون صورت این مثلث در جهت y است، عمودش باید برای واقعی بودن در این جهت باشد؛ یعنی بردار (0,1,0) که در شکل با محور قرمز نشان داده است. به همین شکل این راس در مثلث EAB عمود (-1,0,0) دارد. برای مثلث ACB هم چون زیر هرم است باید عمودش رو به پایین باشد؛ یعنی (0,0,-1).



شکل ۴-۱۴ بردار های عمود اضلاع هرم

برای مثلث EBC در شکل ۴-۱۴ بردار عمود راس E به بالا تمایل دارد، برای مثال فرض کنید کنید بردار عمودش برابر (0,1,1) باشد. با این شرط در نورپردازی مثلث EBC مقدار نور نقطه‌ی وسط B و E برابر الحاق خطی این دو عمود در وسط می‌شود؛ یعنی (0,1,0.5)

در نهایت با تنظیم بردارهای عمود یک آرایه از ۱۸ راس داریم که موقعیت و عمودشان مشخص است. با وارد این کردن این ۱۸ راس به ترتیب درست هرم نهایی قابل رندر شدن در محیط سه‌بعدی است.



شکل ۴–۱۵ نمایش نهایی هرم در محیط سه‌بعدی گرافیکی

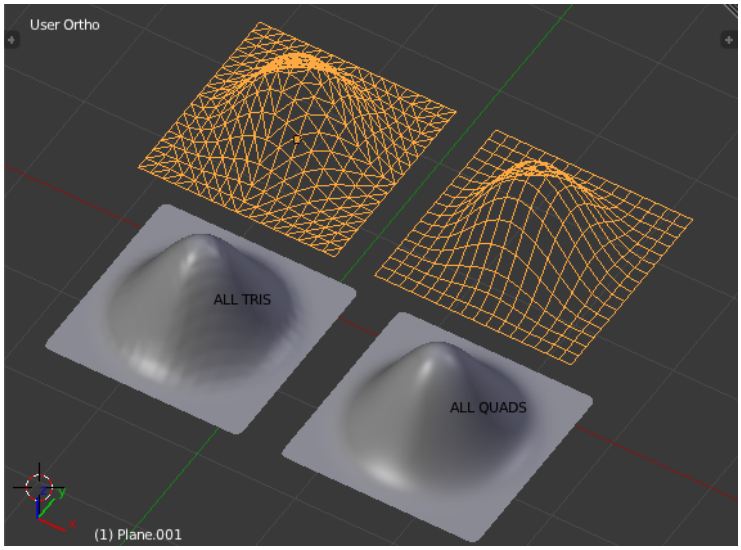
۴-۴-توپولوژی های نمایش توری

نمایش توری می‌تواند توپولوژی‌های مختلفی داشته باشد [۳۰]. در بخش‌های قبلی مثالی از توپولوژی مثلثی دیده شد که یک سطح که کوچکترین واحد قابل دیده شدن در یک شی است توسط سه مثلث تشکیل می‌شود. این توپولوژی می‌تواند بصورت های دیگری نیز باشد که به اختصار به آنها می‌پردازیم.

در اکثر توپولوژی‌های استفاده شده در گرافیک کامپیوتری، توری‌های چندضلعی توسط سطوح بیرونی آنها نمایش داده می‌شود و این به این معناست که داخل آن‌ها خالی است. برخلاف واقعیت که اجسام جامد ضخامت دارند، در این توپولوژی‌ها ساختار حجمی درونی اشیاء بازنمایی نمی‌شود و برای نمایش هر شی به رندر کردن بخش بیرونی آن بسنده می‌شود.

۴-۴-۱-توپولوژی چهارگوش

این توپولوژی مانند توپولوژی مثلث است و بجای سه راس، هر سطح از چهار راس تشکیل شده است. این توپولوژی بیشتر در محیط‌های ویرایش مدل‌های سه‌بعدی مورد استفاده قرار می‌گیرد؛ چرا که به راحتی قابل ویرایش است و اضافه و کم کردن رئوس دسته جمعی بصورت خودکار در این توپولوژی بسیار آسان است. علاوه بر این تبدیل چهارگوش‌ها به سه‌گوش بسیار راحت انجام می‌گیرد، این در حالی است که عملیات برعکس نیازمند صرف زمان زیاد است و به نتیجه قطعی نمی‌رسد.



شکل ۴-۱۶ مقایسه مدل سیمی و نورپردازی شده توپولوژی چهارگوش و مثلثی

۴-۴-۲-توپولوژی ضلعی

در این توپولوژی اضلاع که شامل دو راس هستند ذخیره می‌شوند و در یک لیست دیگر مجموعه‌ای از سه یا چهار ضلع تشکیل سطح می‌دهند.

۴-۴-۳-توپولوژی ابر نقاط

این توپولوژی که برای نمایش به پردازش زیادی نیاز دارد از نقاطی روی سطح یک جسم سه‌بعدی تشکیل می‌شود (شکل ۴-۱۷). این توپولوژی به سختی توسط دست قابل تولید است و با پردازش-هایی تبدیل به توپولوژی‌های راحت‌تر می‌شود.

شکل ۴–۱۷ توپولوژی ابر نقاط برای ساخت یک مدل قوری

یکی از کاربردهای این توپولوژی تبدیل اشیای دنیای واقعی به داده توری هستند که با تکنولوژی تشخیص نوری و تکاوری[1] توسط دوربین‌های مخصوص قابل انجام است.

۴-۵-تفاوت های محیط بلادرنگ

همانطور که در بخش قبل گفته شد، محدودیت زمانی برای محیط‌های گرافیکی بلادرنگ مسئله‌ی بسیار مهمی است. برای همین در این محیط‌های تغییراتی صورت می‌گیرد که به کاهش زمان نمایش اشیاء روی صفحه نمایش کمک کند. در این بخش به سه تکنیک مخصوص محیط‌های بلادرنگ می‌پردازیم.

۴-۵-۱-جمع‌آوری سطوح پشتی[2]

در این تکنیک که در اکثر موتورهای بازی‌سازی بصورت پیشفرض انجام می‌شود، تمامی سطوح یک طرفه هستند؛ به این معنی پشت آنها روی صفحه نمایش رندر نمی‌شود.

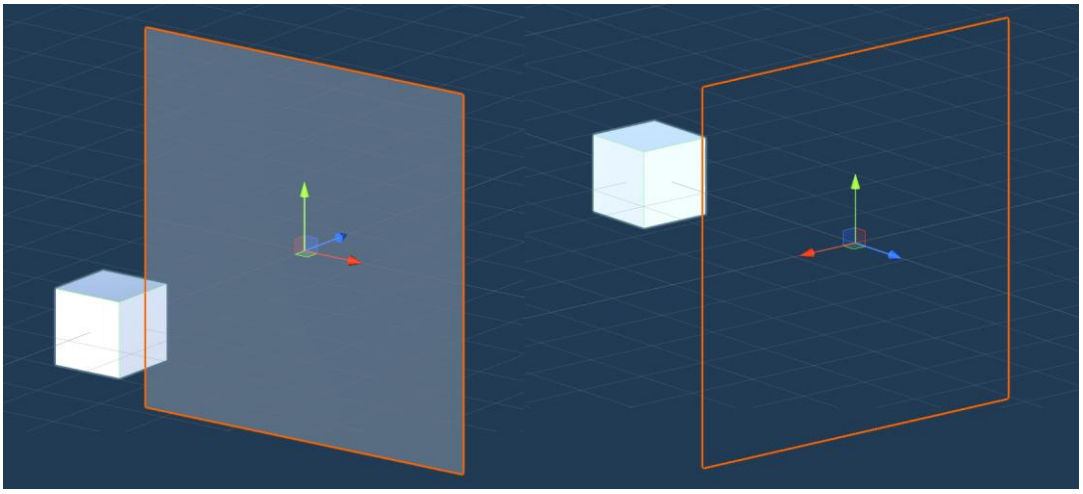
اشیایی که از صفحات در موتورها ساخته شده‌اند مثال خوبی برای نمایش این مورد هستند. همچنین با ورود دوربین به داخل اشیاء، دیده می‌شود که سطح داخلی این اشیاء نمایش داده نمی‌شود. این باعث می‌شود برای پرکردن داخل یک جسم، نیاز به محاسبات اضافی داشته باشیم.

[1] Light detection and ranging (Lidar)
[2] Back-culling

در اینجا لازم است توضیحاتی درباره چگونگی انجام این کار ارائه شود؛ چرا که از این مفهوم در فصل‌های بعدی استفاده خواهد شد. در محیط موتورهای بازی‌سازی، توری‌ها از مثلث‌ها تشکیل شده‌اند اما تعیین اینکه کدام سمت یک مثلث در صفحه نمایش داده شود به ترتیب قرارگیری اندیس‌های آن در لیست مربوط به اندیس‌ها مربوط می‌شود.

بصورت قراردادی، فقط مثلث‌هایی که رئوس آن بصورت ساعتگرد مرتب شده است رو به دوربین نمایش داده می‌شود. به همین شیوه پشت یک مثلث بصورت برعکس خواهد بود و نمایش داده نخواهد شد. این نکته‌ای مهم در دستکاری توری می‌باشد چرا که با تغییر جهت اندیس‌های یک مثلث، شکل نهایی با شکل مورد انتظار متفاوت خواهد بود.

در شکل ۴-۱۸ یک صفحه از روبرو (سمت چپ) و پشت (سمت راست) دیده می‌شود. همانطور که دیده می‌شود از پشت قادر به دیدن صفحه نیستیم.



شکل ۴-۱۸ جمع‌آوری سطوح پشتی یک صفحه

۴-۵-۲-جمع‌آوری اشیای مسدود شده[1]

این تکنیک اشیایی که از مخروط دوربین بیرون هستند و یا توسط شی دیگری مسدود شده‌اند توسط کارت گرافیکی رندر نمی‌شود [۳۱] که زمان کوتاهتری برای رندر سپری شود. این روش می‌تواند به تکنیک پرتاب اشعه انجام شود.

[1] Occlusion-culling

۴-۵-۳-صفحه‌ی دور دوربین

این صفحه در دوربین تعیین می‌کند تا که عمقی دوربین سعی در نمایش دادن اشیای موجود در محیط بکند. برای مثال اگر صفحه‌ی دور دوربین ۱۰ متر باشد، اشیایی که در مخروط دوربین هستند ولی در فاصله بیش از ۱۰ متری از آن در جهت بردار عمق محلی دوربین هستند، در رندر نهایی نمایش داده نمی‌شود.

۴-۶-انواع توری‌های سه‌بعدی

چندضلعی‌های سه بعدی می‌توانند محدب یا مقعر باشند. (شکل ۴-۱۹) در یک دسته‌بندی دیگر این چندضلعی‌ها می‌توانند ساده یا غیرساده باشند (شکل ۴-۲۰) [۳۲] که در این بخش به معرفی آنها می‌پردازیم.

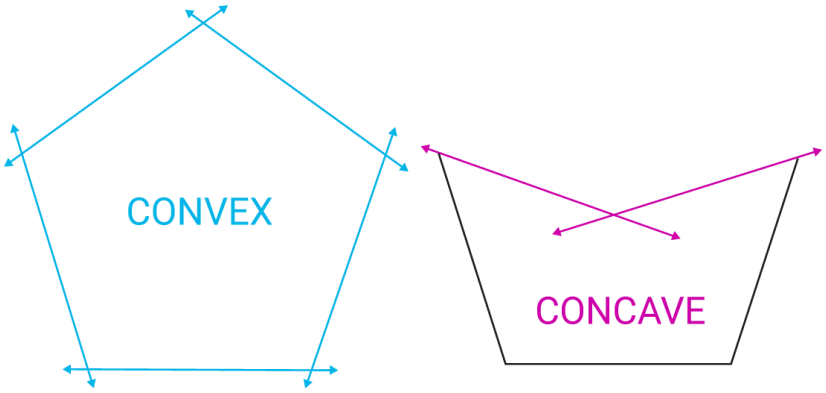
لازم به ذکر است که توری‌های محدب پس از برش، همواره چندضلعی محدب تشکیل می‌دهند و توری‌های مقعر پس از برش ممکن است چندضلعی‌های مقعر تشکیل دهند.

۴-۶-۱-چندضلعی‌های محدب

در چندضلعی‌های محدب، نمی‌توان هیچ دو راسی یافت که ضلع تشکیل دهنده‌ی آن دو بیرون آن چند ضلعی قرار گیرد. زاویه درونی رئوس در این چند ضلعی‌ها حداکثر ۹۰ درجه است.

۴-۶-۲-چندضلعی‌های مقعر

به چند ضلعی‌ای که حداقل یک جفت راس بتوان در آن یافت که ضلع تشکیل دهنده آن دو از محیط تشکیل دهنده آن چند ضلعی بیرون بیفتد، چند ضلعی مقعر می‌گوییم. زاویه درونی حداقل یک راس در این چندضلعی‌ها از ۹۰ بیشتر است.



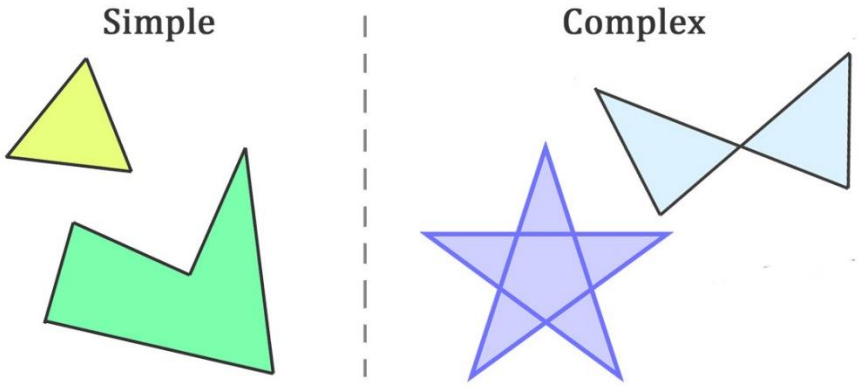
شکل ۴–۱۹ تفاوت چند ضلعی‌های محدب و مقعر [۳۳]

۴-۶-۳-چندضلعی ساده

در یک چندضلعی ساده رئوس پشت سر هم با یک ضلع به هم وصل می‌شوند و راس آخر به یک ضلع به راس اول وصل می‌شود. در این چند ضلعی‌ها تنها نقطه برخورد اضلاع، روی گوشه‌های این چندضلعی است.

۴-۶-۴-چندضلعی غیرساده

به هر چند ضلعی که ساده نباشد، غیرساده می‌گوییم. برای مثال یک ضلع می‌تواند با ضلع دیگری برخورد داشته باشد و یا بیش از دو ضلع از یک راس گذر کرده باشند.



شکل ۴–۲۰ تفاوت چندضلعی ساده و پیچیده (غیرساده) [۳۲]

۴-۶-۵-چندضلعی سوراخ دار

به چندضلعی که درون خود یک چند ضلعی دیگر دارد که سطوح درونش پر نشده است، چند ضلعی سوراخ دار می‌گوییم. در این پایان‌نامه از آنجایی که محاسبه مثلثی سازی چندضلعی‌های سوراخ دار پیچیدگی زیادی دارد، به این چندضلعی‌ها پرداخته نمی‌شود.

فصل ۵

الگوریتم‌های برش توری سه‌بعدی

الگوریتم‌های برش توری سه‌بعدی

در این فصل به الگوریتم‌های اصلی استفاده شده جهت برش یک شی سه‌بعدی می‌پردازیم و جزئیات ریاضیاتی آن را واکاوی می‌کنیم.

۵-۱-بازنمایی صفحه در محیط سه‌بعدی

هر صفحه سه‌بعدی را می‌توان به چندین شکل مختلف بازنمایی کرد. برای سهولت محاسبات الگوریتم، در اینجا هر صفحه را با یک بردار عمود n و یک عدد d نشان می‌دهیم.

در این انتزاع صفحه، بردار عمود n نشان دهنده جهت صفحه است و بر تمامی نقاط درون صفحه عمود است. d بیانگر فاصله‌ی نقاط از بردار عمود است؛ به عبارت دیگر طولی که باید در جهت بردار n طی کنیم تا به صفحه موردنظر برسیم.

به طور ساده‌تر می‌توان یک صفحه را به سه نقطه بازنمایی کرد. در معادلات ۵-۱ و ۵-۲ تبدیل این انتزاع به انتزاع بردار عمود و فاصله را نشان می‌دهیم.

$$\vec{n} = (\overrightarrow{p_2} - \overrightarrow{p_1}) \times (\overrightarrow{p_3} - \overrightarrow{p_1}) \qquad \text{معادله ۵-۱}$$

$$d = \overrightarrow{p_1} . \vec{n} \qquad \text{معادله ۵-۲}$$

همچنین روش یک نقطه و بردار عمود برای بازنمایی قابل فهم‌تر است. در این روش هم فاصله d مانند فرمول بالا از ضرب داخلی نقطه روی صفحه و بردار عمود بدست می‌آید.

در اینجا لازم است یادآوری کنم که این انتزاع از صفحه سه‌بعدی، یک صفحه‌ی نامحدود در جهت طول و عرض محلی را بازنمایی می‌کند؛ عملیات محدودسازی صفحه در پیش‌پردازش‌ها انجام می‌شود که به تفصیل در فصل ششم توضیح داده خواهد شد.

۵-۲-بازنمایی توری در محیط سه بعدی

در اینجا فرض می‌کنیم هر توری از مثلث‌هایی تشکیل شده است که هر مثلث سه راس دارد. برای سادگی فرض می‌کنیم هر راس فقط شامل داده مکانی، یو وی و عمود خود است. از روش‌های دیگری از جمله ابر نقطه برای برش محاسبات ساده‌تری دارند که به دلیل اینکه بخاطر کارایی کم در زمینه بازی‌های ویدئویی استفاده نمی‌شود و نیازمند پس‌پردازش سنگین هستند، صرف‌نظر می‌شود.

۵-۳-کاهش بعد

یک روش که چندین بار از آن استفاده می‌کنیم تصویر یک نقطه‌ی سه‌بعدی به دوبعدی است که در اینجا لازم است توضیح داده شود. برای این کار فرض کنید می‌خواهیم نقطه p را روی صفحه‌ای تصویر کنیم. بردار عمود این صفحه چیزی است که حائز اهمیت است؛ چون در تصویر کردن اینکه صفحه در چه مکانی باشد اهمیتی ندارد و تا وقتی که بردار عمود یکسان باشد، خروجی یکسان می‌گیریم.

برای این کار کافی است با توجه به بردار عمود، دو بردار پیدا کنیم که محورهای دو بعد را تشکیل می‌دهند. این دو محور باید دو ویژگی زیر را داشته باشند:

- بر هم عمود باشند
- بر بردار عمود، عمود باشند

برای این کار می‌توان یک بردار دلخواه a را در نظر گرفت که برابر بردار عمود صفحه نباشد. بردارهای مورد نظر مطابق با معادلات ۵-۳ و ۵-۴ محاسبه می‌شوند.

$$\vec{u} = \left\|\overrightarrow{normal} \times \vec{a}\right\| \qquad \text{معادله ۵-۳}$$

$$\vec{v} = \left\|\vec{u} \times \overrightarrow{normal}\right\| \qquad \text{معادله ۵-۴}$$

حال با داشتن این دو محور می‌توان هر نقطه‌ی p را با معادله ۵-۵ در دو بعد بدست آورد.

$$p_{projected} = (\vec{p}.\vec{u}, \vec{p}.\vec{v}) \qquad \text{معادله ۵-۵}$$

۵-۴-برش یک مثلث در یک توری

برای فهم راحت‌تر الگوریتم برش، از واحد سطح[1] شروع می‌کنیم، در نهایت هر توری از چندین مثلث تشکیل شده است و تکرار این کار موجب برش توری می‌شود که در بخش بعدی توضیح داده خواهد شد.

[1] Face

در نهایت خروجی یک برش دو توری جدید است که در اینجا از آنها با نام توری بالایی و پایینی یاد می‌کنیم. توری بالایی در بالای صفحه برش قرار دارد و توری پایینی در پایین آن قرار می‌گیرد.

یادآوری این نکته لازم است که در بسیاری از موتورهای بازی‌سازی از جمله یونیتی، مثلث‌های رو به دوربین نمایش داده می‌شود که رئوسش بصورت ساعتگرد مرتب شده باشند؛ از این رو در این بخش باید دقت داشت که ترتیب ایجاد مثلث‌های جدید بهم نریزد.

در اینجا نیاز است از انتزاعی که از صفحه ساختیم استفاده نماییم. برای تشخیص موقعیت یک راس v نسبت به یک صفحه p می‌توان از معادله ۵-۶ استفاده کرد.

$$relation(v,p) = \begin{cases} top, & (v_{position} \cdot p_{normal}) > p_{distance} \\ surface, & (v_{position} \cdot p_{normal}) = p_{distance} \\ button, & (v_{position} \cdot p_{normal}) < p_{distance} \end{cases}$$

معادله ۵-۶

همانطور که دیده می‌شود نسبت یک راس می‌تواند با یک صفحه سه نوع باشد. یک راس می‌تواند در بالای صفحه قرار گیرد، روی صفحه باشد و یا در زیر آن قرار گرفته باشد.

همچنین برای محاسبه نقطه تقاطع یه ضلع مثلث با یه صفحه، می‌توان معادله ضلع ab را بصورت معادله الحاق خطی نوشت (معادله ۵-۷).

$$edge(a,b) = (\vec{b} \times t) + \vec{a} \times (1-t)$$

معادله ۵-۷

در فرمول بالا t می‌تواند بین صفر و یک باشد. برای محاسبه نقطه تقاطع ab با صفحه p کافی است t را طوری تنظیم کنیم که روی صفحه‌ی p بیفتد. برای این کار می‌توان به شکل زیر عمل نمود.

ابتدا با ضرب داخلی a در بردار عمود صفحه p، نسبت فاصله a به صفحه p را بدست می‌آوریم. (معادله ۵-۸)

$$a_{distance} = \overrightarrow{p_{normal}} . \overrightarrow{a_{position}}$$ معادله ۵-۸

سپس با تفریق بردار a از b، بردار خالص ab را بدست می‌آوریم. (معادله ۵-۹)

$$\overrightarrow{ab} = \vec{b} - \vec{a}$$ معادله ۵-۹

طبق معادله ۵-۱۰، با تکرار ضرب داخلی ab که در معادله ۵-۹ محاسبه شد، روی بردار عمود صفحه، می‌توان طول بردار ab را که روی عمود صفحه p نگاشت شده است، بدست آورد.

$$\overrightarrow{ab_{mapped\ length}} = \overrightarrow{p_{normal}} . \overrightarrow{ab}$$ معادله ۵-۱۰

در نهایت با نسبت گرفتن فاصله a نسبت به صفحه p، به طول بردار نگاشت شده ab روی عمود صفحه p، t بدست می‌آید. (معادله ۵-۱۱)

$$t_{intersection} = \frac{p_{distance} - a_{distance}}{ab_{mapped\ length}}$$ معادله ۵-۱۱

با جاگذاری $t_{intersection}$ در معادله ۵-۷ می‌توان نقطه‌ی برخورد ضلع ab و صفحه p را بدست آورد.

بصورت کلی برخورد یک صفحه با یک مثلث به روش‌های زیادی انجام می‌شود که در ادامه به آنها پرداخته می‌شود. برای ثبات رئوس مثلث با a، b و c نشان داده می‌شوند و جهت abc ساعتگرد فرض می‌شود.

حالت اول: سه راس مثلث در وضعیت یکسانی از صفحه قرار گرفته باشند؛ در این حالت هیچ تقاطعی صورت نمیگیرد و عملیات دو نیم‌سازی انجام نمی‌گیرد و مثلث به همان حالت قبل باقی می‌ماند.

- سه راس مثلث در بالای صفحه قرار گیرند؛ در این حالت این مثلث abc متعلق به توری بالایی است.
- سه راس مثلث پایین صفحه قرار گیرند؛ در این حالت مثلث abc به توری پایینی اضافه می‌شود.
- سه راس مثلث روی صفحه قرار گیرند؛ در این حالت مثلث abc به هر دو توری اضافه می‌شود.

حالت دوم: فقط دو راس مثلث روی صفحه قرار گیرند

- راس سوم بالای صفحه قرار گیرد؛ در این حالت مثلث abc به توری بالایی تعلق می‌گیرد.
- راس سوم پایین صفحه قرار گیرد؛ در این حالت مثلث abc به توری پایینی تعلق می‌گیرد.

حالت سوم: فقط یک راس مثلث روی صفحه قرار گیرد

- فقط راس a روی صفحه گیرد
  - دو راس دیگر در بالای صفحه باشند؛ در این حالت مثلث abc به توری بالایی تعلق می‌گیرد.
  - دو راس دیگر در پایین صفحه باشند؛ در این حالت مثلث abc به توری پایینی تعلق می‌گیرد.
  - راس b بالا و راس c پایین صفحه باشند؛ در این حالت بین ضلع bc و صفحه تقاطع i ایجاد می‌شود و دو مثلث abi و aic ایجاد می‌شوند. مثلث abi به توری بالایی و مثلث aic به توری پایینی اضافه می‌شود.
  - سه. یک. چهار: راس c بالا و راس b پایین صفحه باشند؛ در این حالت بین ضلع bc و صفحه تقاطع i ایجاد می‌شود و دو مثلث abi و aic ایجاد می‌شوند. مثلث abi به توری پایینی و مثلث aic به توری بالایی اضافه می‌شود.
- فقط راس b روی صفحه گیرد
  - دو راس دیگر در بالای صفحه باشند؛ در این حالت مثلث abc به توری بالایی تعلق می‌گیرد.

- دو راس دیگر در پایین صفحه باشند؛ در این حالت مثلث abc به توری پایینی تعلق می‌گیرد.
- راس a بالا و راس c پایین صفحه باشند؛ در این حالت بین ضلع ac و صفحه تقاطع i ایجاد می‌شود و دو مثلث bia و bci ایجاد می‌شوند. مثلث bia به توری بالایی و مثلث bci به توری پایینی اضافه می‌شود.
- راس c بالا و راس a پایین صفحه باشند؛ در این حالت بین ضلع ac و صفحه تقاطع i ایجاد می‌شود و دو مثلث bia و bci ایجاد می‌شوند. مثلث bia به توری پایینی و مثلث bci به توری بالایی اضافه می‌شود.

- فقط راس c روی صفحه گیرد
  - دو راس دیگر در بالای صفحه باشند؛ در این حالت مثلث abc به توری بالایی تعلق می‌گیرد.
  - دو راس دیگر در پایین صفحه باشند؛ در این حالت مثلث abc به توری پایینی تعلق می‌گیرد.
  - راس b بالا و راس a پایین صفحه باشند؛ در این حالت بین ضلع ab و صفحه تقاطع i ایجاد می‌شود و دو مثلث cai و cib ایجاد می‌شوند. مثلث cai به توری بالایی و مثلث cib به توری پایینی اضافه می‌شود.
  - راس a بالا و راس b پایین صفحه باشند؛ در این حالت بین ضلع ab و صفحه تقاطع i ایجاد می‌شود و دو مثلث cai و cib ایجاد می‌شوند. مثلث cai به توری پایینی و مثلث cib به توری بالایی اضافه می‌شود.

حالت چهارم: هیچ راسی روی صفحه نباشد و در وضعیت یکسانی قرار نگرفته باشند

- راس a تنها باشد
  - راس a در زیر صفحه باشد؛ در این حالت ab و ac تقاطع i و i’ را ایجاد می‌کنند و سه مثلث aii’، ibi’ و i’bc ایجاد می‌شود. در این سه مثلث مثلث اول به توری پایینی اضافه می‌شود و دو مثلث دیگر به توری بالایی اضافه می‌شوند.
  - راس a در بالای صفحه باشد؛ در این حالت ab و ac تقاطع i و i’ را ایجاد می‌کنند و سه مثلث aii’، ibi’ و i’bc ایجاد می‌شود. در این سه مثلث مثلث اول به توری بالایی اضافه می‌شود و دو مثلث دیگر به توری پایینی اضافه می‌شوند.

- راس b تنها باشد
  - راس b در زیر صفحه باشد؛ در این حالت ab و bc تقاطع i و 'i را ایجاد می‌کنند و سه مثلث bi'i، i'ci و ica ایجاد می‌شود. در این سه مثلث مثلث اول به توری پایینی اضافه می‌شود و دو مثلث دیگر به توری بالایی اضافه می‌شوند.
  - راس b در بالای صفحه باشد؛ در این حالت ab و bc تقاطع i و 'i را ایجاد می‌کنند و سه مثلث bi'i، i'ci و ica ایجاد می‌شود. در این سه مثلث مثلث اول به توری بالایی اضافه می‌شود و دو مثلث دیگر به توری پایینی اضافه می‌شوند.
- راس c تنها باشد
  - راس c در زیر صفحه باشد؛ در این حالت ac و bc تقاطع i و 'i را ایجاد می‌کنند و سه مثلث ci'i، ai'i و abi' ایجاد می‌شود. در این سه مثلث مثلث اول به توری پایینی اضافه می‌شود و دو مثلث دیگر به توری بالایی اضافه می‌شوند.
  - راس c در بالای صفحه باشد؛ در این حالت ac و bc تقاطع i و 'i را ایجاد می‌کنند و سه مثلث ci'i، ai'i و abi' ایجاد می‌شود. در این سه مثلث مثلث اول به توری بالایی اضافه می‌شود و دو مثلث دیگر به توری پایینی اضافه می‌شوند.

۵-۵-برش یک توری

برای برش یک توری در نهایت می‌توان روی تمامی مثلث‌ها یک به یک الگوریتم برش یک مثلث را اجرا کرد تا خروجی مورد نظر تولید شود.

با وجود اینکه موتور بازی‌سازی یونیتی هنوز از پردازش چندهسته‌ای بصورت بومی پشتیبانی نمی‌کند، این حلقه به راحتی قابل موازی سازی است ولی با توجه به تعداد کم مثلثات در یک توری در بازی‌های ویدئویی و پیاده‌سازی کارا در فصل شش و آزمایشات انجام شده در فصل هفت نیاز به این کار نمی‌باشد.

همانطور که قبلا گفته شد، هر مثلث شامل داده‌های دیگری از جمله یو وی و عمود هم هست. راس‌هایی که هیچ تغییری نداشتند علاوه بر موقعیت، یو وی و عمود آنها نیز تغییری نمی‌کند ولی راس‌های جدید ایجاد شده، که در بخش قبل با i نشان داده شد، باید یو وی و عمود آن‌ها نیز تولید شود که برای این کار روش‌های زیادی وجود دارد.

فرض کنید راس جدید i بین رئوس a و b قرار دارد و با صفحه بُرنده تقاطع داشته است. همانطور که در بخش قبل گفته شد، طبق معادله ۵-۱۱ می‌توان $t_{intersection}$ را بدست آورد. از همین t می‌توان بردار عمود و یو وی را بدست آورد.

برای بدست آوردن بردار عمود راس i، کافی است میانگین وزن دار بردار عمود a و b را بدست آوریم. وزن مذکور توسط t تعیین می‌شود. مقدار t برابر یک، یعنی همه‌ی وزن میانگین به b داده می‌شود؛ این منطقی است، چرا که اگر t یک باشد، یعنی نطقه تقاطع دقیقا روی b قرار دارد. به همین شکل با داشتن t صفر تمام وزن به a تعلق می‌گیرد.

نکته قابل توجه اینجاست که برای اطمینان از خراب نشدن نورپردازی در بردار عمود، پس از محاسبه میانگین وزن دار، بردار بدست آمده را نرمال کنیم؛ با این کار مطمئن می‌شویم طول این بردار ۱ است و در معادله ۵-۱۲ شدت نور از یک عبور نمی‌کند.

$$i_{normal} = \|a_{normal} \times (1-t) + b_{normal} \times t\| \qquad \text{معادله ۵-۱۲}$$

برای محاسبه یو وی نیز به همین شکل عمل می‌شود. با توجه به اینکه بردار یو وی، یک بردار نگاشت دوبعدی است و تمام رئوس یو وی بین (۰،۰) و (۱،۱) دارند، میانگین وزن دار دو نقطه به هیچ وجه از این بازه تخطی نمی‌کند و همواره معتبر خواهد بود. (معادله ۵-۱۳)

$$i_{uv} = a_{uv} \times (1-t) + b_{uv} \times t \qquad \text{معادله ۵-۱۳}$$

۵-۶-توپر کردن سطح برش

توپر کردن سطح برش با مثلثی سازی روی رئوس جدید ایجاد شده انجام می‌شود. این مسئله یکی از مسائل مهم گرافیک کامپیوتری است و با توجه به نوع سطح ایجاد شده می‌تواند پیچیدگی‌های مختلفی داشته باشد. در این پروژه فرض می‌کنیم قرار است توری‌های محدب ساده را برش دهیم.

شایان ذکر است که میتوان یک توری مقعر را به چند توری محدب، تبدیل کرد و از الگوریتم قشر محدب برای مثلثی‌سازی کمک گرفت. در ادامه به معرفی الگوریتم‌های قشر محدب می‌پردازیم، به علاوه یک روش دیگر برای حل مثلثی‌سازی چندضلعی‌های محدب نیز معرفی می‌شود.

برای توپر سازی یک چندضلعی ابتدا باید ترتیب تشکیل مثلث‌ها را بدست آوریم که این کار با الگوریتم قشر محدب انجام می‌شود و در مرحله بعد باید مثلثی‌سازی را با توجه به خروجی این قسمت انجام دهیم.

قبل از اینکه الگوریتم‌های اصلی موجود را بررسی کنیم باید مسئله‌ی را تعریف کنیم؛ این مسئله بصورت زیر تعریف می‌شود:

با داشتن مجموعه‌ای از نقاط S روی یک صفحه، مجموعه‌ای از نقاط را بیابید که از شکل‌گیری یک چندضلعی از آن، تمام نقاط S داخل آن باشند و هیچ نقطه‌ای بیرون آن قرار نگیرد.

برای مثلثی سازی توری‌های ساده، عملا نقاط دیگری موجود نیستند و از تمام رئوس برای ساخت قشر بیرونی استفاده می‌شود. برای همین باید به این نکته توجه کرد که بعضی از الگوریتم‌ها برای این کار اصلا مناسب نیستند؛ چرا که با این فرض طراحی شده‌اند که نقاط زیادی درون چندضلعی قرار می‌گیرد.

۵-۶-۱-الگوریتم پیمایش جارویس[1] [۳۴]

این الگوریتم که با نام کادوپیچی هم شناخته می‌شود، در سال ۱۹۷۳ توسط ری جارویس[2] منتشر شد. پیچیدگی زمانی این الگوریتم $O(nh)$ است که n نشان دهنده کل نقاط و h نشانگر تعداد نقاط روی قشر بیرونی است. این الگوریتم برای مواقعی که n و h با هم برابر باشند (مثلا در مسئله مثلثی سازی) به پیچیدگی $O(n^2)$ تبدیل می‌شود که برای کاربردهای بلادرنگ مناسب نیست.

[1] Jarvis March
[2] R. A. Jarvis

این الگوریتم با چپ ترین نقطه شروع می‌کند که مسلما روی قشر بیرونی قرار دارد. در ادامه در جهت خلاف عقربه‌های ساعت (یا برعکس) پیشروی می‌کند و با پیمایش نقاط چندضلعی، راسی را پیدا می‌کند که زاویه تشکیل دهنده آخرین نقطه پیدا شده در قشر با آن نسبت به امتداد دو نقطه آخر موجود در قشر از سایرین کمتر باشد، این کار تا زمانی ادامه می‌یابد که نقطه‌ی پیدا شده، برابر با نقطه‌ی اول، یعنی چپ‌ترین نقطه باشد، و با رسیدن به این حالت، پیچیدن کادو تمام می‌شود و الگوریتم به پایان می‌رسد.

خروجی این الگوریتم به ترتیب ساعتگرد یا پادساعتگرد می‌باشد که برای استفاده جهت مثلثی سازی مناسب است.

Algorithm 1: JARVIS MARCH ALGORITHM

```
    Input: a list S of points in a plane
    Output: a list P of points which forms S convex hull
1   pointOnHull = leftmost point in S
2   i := 0

3   repeat
4       P[i] := pointOnHull
5       endpoint := S[0]
6     for j from 0 to |S| do
7           if (endpoint == pointOnHull) or (S[j] is on left of line from P[i] to
    endpoint) then
8               endpoint := S[j]
9       i := i + 1
10      pointOnHull = endpoint
11  until endpoint = P[0]
12  return P
```

این الگوریتم قادر به محاسبه‌ی قشر بیرونی در ابعاد بالاتر نیز است، که در این مسئله مورد نیاز ما نیست؛ چرا که می‌توان نقاط سه‌بعدی ایجاد شده توسط تقاطع‌ها را روی صفحه‌ی برنده نگاشت کرد و به تصویر دوبعدی از آن دست یافت.

۵-۶-۲-الگوریتم زنجیره مونوتون[1] [۳۵]

این الگوریتم با مرتبه زمانی O(nlogn) می‌تواند عملیات حل قشر بیرونی را برای چندضلعی‌های محدب انجام دهد. این الگوریتم در ابتدا مجموعه نقاط ورودی را مرتب می‌کند (که گلوگاه این الگوریتم است) و سپس در O(n) دو زنجیره بالایی و پایینی تشکیل می‌دهد که با پردازش روی آرایه‌های تشکیل شده می‌توان به ترتیب پادساعتگرد از لیست اجزای قشر بیرونی رسید.

Algorithm 2: ANDREW'S MONOTONE CHAIN CONVEX HULL ALGORITHM

```
    Input: a list S of points in the plane.
    Output: a list P of points which forms S convex hull
1   Sort the points of S by x-coordinate (in case of a tie, sort by y-coordinate).
2   Initialize U and L as empty lists.
3   for i = 1, 2, ..., n:
4       while L contains at least two points and the sequence of last two points
            of L and the point S[i] does not make a counter-clockwise turn:
5           remove the last point from L
6       append S[i] to L

7   for i = n, n-1, ..., 1:
8       while U contains at least two points and the sequence of last two points
            of U and the point S[i] does not make a counter-clockwise turn:
9           remove the last point from U
10      append S[i] to U

11
12  Remove the last point of each list
    P := Concatenate L and U to obtain the convex hull of S.
13  return P
```

این الگوریتم به این شکل عمل می‌کند که دو قشر بالایی و پایینی را می‌سازد و در نهایت پس از حذف نقاط آخر آنها، آنها را به هم متصل می‌کند. حذف نقطه‌ی آخر به این دلیل اتفاق می‌افتد که نقطه‌ی آخر هر قشر، نقطه‌ی اول قشر دیگر است.

[1] Monotone Chain

۵-۶-۳-الگوریتم اسکن گراهام[1] [۳۶]

این الگوریتم که توسط رونالد گراهام[2] در سال ۱۹۷۲ معرفی شد، می‌تواند با پیچیدگی زمانی O(nlogn) قشر بیرونی یک چندضلعی محدب را پیدا کند. این الگوریتم با به کار گیری ساختار داده پشته قشر بیرونی یک چندضلعی را نگه‌داری می‌کند.

در مرحله اول این الگوریتم پایین‌ترین نقطه چندضلعی پیدا می‌شود که به O(n) زمان نیاز دارد. در مرحله بعد که گلوگاه این الگوریتم است باید عملیات مرتب‌سازی انجام شود که بر اساس زاویه پایین‌ترین نقطه با رئوس نسبت به محور x انجام می‌شود. این مرحله O(nlogn) زمان می‌برد.

در نهایت با توجه به مرتب‌سازی انجام شده نقاط به ترتیب وارد پشته می‌شوند و تا وقتی که همواره در جهت پادساعتگرد حرکت شود (مانند الگوریتم زنجیره مونوتون) این ترتیب ادامه دارد و با رسیدن به یک عضو که باعث ایجاد چرخش ساعتگرد می‌شود، آخرین عنصر از پشته خارج می‌شود.

این الگوریتم در بدترین شرایط می‌تواند در nlogn به خروجی برسد.

Algorithm 3: GRAHAM SCAN ALGORITHM

```
    Input: a list S will be the list of points
    Output: a list P of points which forms S convex hull
1   stack := empty_stack()
2   Find the lowest y-coordinate and leftmost point, called P0
3   Sort points by polar angle with P0, if several points have the same polar angle
    then only keep the farthest

4   for point in points:
5      while count stack > 1 and ccw(next_to_top(stack), top(stack), point) <=
    0:
6          pop stack
7      push point to stack
8   end
9
10  P := convert stack to array
11  return P
```

[1] Graham Scan
[2] Ronald Graham

در سه الگوریتم معرفی شده همواره نیاز به محاسبه جهت چرخش راس‌ها داشتیم. معادله ۵-۱۴ نحوه بدست آوردن آن را نشان می‌دهد. این فرمول با سه نقطه تعیین می‌کند که آیا نقطه سوم در سمت چپ امتداد نقطه اول و دوم قرار دارند (چرخش پادساعتگرد) یا راست (چرخش ساعتگرد) یا در امتداد هم هستند.

$$ccw(a,b,c) = \begin{cases} clockwise & (b_x - a_x) \times (c_y - a_y) - (b_y - a_y) \times (c_x - a_x) < 0 \\ counterclockwise & (b_x - a_x) \times (c_y - a_y) - (b_y - a_y) \times (c_x - a_x) > 0 \\ colinear & (b_x - a_x) \times (c_y - a_y) - (b_y - a_y) \times (c_x - a_x) = 0 \end{cases}$$

معادله ۵-۱۴

۵-۶-۴-اثبات کران پایین پیچیدگی الگوریتم های قشر محدب

همانطور که دیده شد، الگوریتم‌های قشر محدب قادر به رسیدن به پیچیدگی زمانی خطی نیست. [۳۷] برای اثبات از روش برهان خلف، فرض کنیم الگوریتمی وجود دارد که در بدترین حالت پیچیدگی زمانی کمتری از nlogn دارد.

با این فرض، اگر یک مجموعه نقطه داشته باشیم ($x_1, x_2, \ldots, x_n$) و آن‌ها را روی یک سهمی قرار دهیم، به این شکل که نقاط نهایی به شکل (x_i, x_i^2) خواهند بود. اگر این مجموعه نقاط دو بعدی مجموعه نقاط ورودی باشند، این الگوریتم قشر محدب این مجموعه را پیدا می‌کند.

این یعنی که این الگوریتم می‌تواند عملیات مرتب سازی این مجموعه نقاط را در زودتر از nlogn انجام دهد که با کران پایین پیچیدگی مرتب‌سازی در تناقض است.

بنابراین نمی‌توان الگوریتمی یافت که قشر محدب یک مجموعه نقاط مرتب نشده روی صفحه را در کمتر از nlogn بدست آورد.

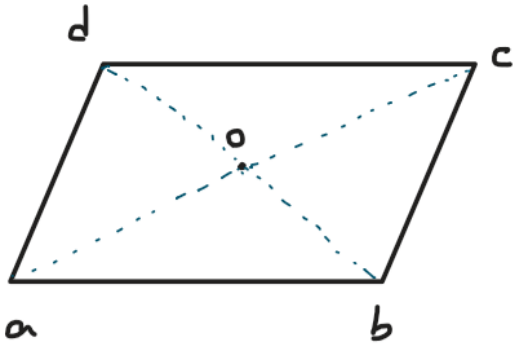
۵-۶-۵-مثلثی سازی قشر محدب

مثلثی‌سازی یک چندضلعی تجزیه‌ی سطح یک چندضلعی به مجموعه‌ای از مثلث هاست. [۳۸] پس از بدست آوردن قشر محدب می‌توان به مثلثی‌سازی آن پرداخت. در این بخش دو توپولوژی ساخت سطح با استفاده از قشر محدب مطرح می‌شود.

مثلث پنکه‌ای مرکزی

در این روش میانگین نقاط قشر محدب به عنوان مرکز پنکه تعیین می‌شود و مثلث‌هایی با کمک این مرکز ساخته می‌شود. راس اول هر مثلث مرکز چندضلعی است و سایر راس‌ها به ترتیب قشر تعیین می‌شوند.

برای مثال اگر چهار راس a، b، c و d داشته باشیم و o مرکز مربع باشد، مثلث ها به شکل oab، obc، ocd و oda خواهند بود. (شکل ۵-۱)

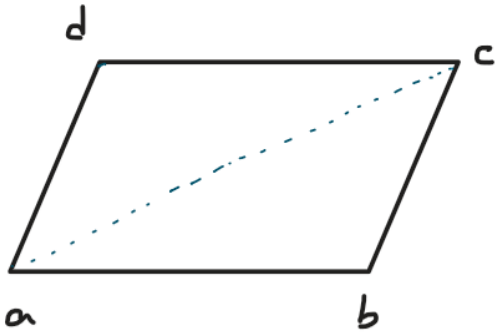


شکل ۵-۱ مثلثی‌سازی پنکه‌ای مرکزی

مثلث پنکه‌ای حاشیه‌ای

در این روش که توپولوژی کثیف‌تری دارد نیاز به ساخت هندسه جدید نیست. یکی از رئوس به اختیار انتخاب می‌شود و مثلث‌ها مانند روش پنکه‌ای منتها با مرکزیت این راس ساخته می‌شوند.

برای مثال اگر چهار راس a، b، c و d داشته باشیم، مثلث ها به شکل abc و acd خواهند بود. (شکل ۵-۲)



شکل ۵-۲ مثلثی‌سازی پنکه‌ای حاشیه‌ای

۵-۶-۶-الگوریتم گوش‌بُری[1] [۳۹]

الگوریتم گوش‌بُری با استفاده از قضیه دو گوش [۴۰]، یکی از روش‌های مثلثی سازی را به طور مستقیم در $O(n^2)$ انجام می‌دهد. در روش‌های معرفی شده قبلی نیاز داشتیم تا پس از بدست آوردن نقاط روی قشر محدب، مثلثی سازی را انجام دهیم ولی این روش مخصوص مثلثی‌سازی طراحی شده است و این مشکل را ندارد.

مشکلی که این الگوریتم را برای انتخاب نامطمئن می‌کند این است که این الگوریتم پیچیدگی زمانی $O(n^2)$ دارد که موجب عدم صلاحیت آن برای یک الگوریتم همه‌منظوره می‌کند.

طبق قضیه دو گوش، هر چندضلعی ساده که دارای بیش از سه راس است، از دو گوش تشکیل شده است. با حذف این گوش‌ها می‌توان دوباره به چندضلعی جدیدی رسید که هنوز ساده است. با تکرار این کار می‌توان به یک مثلث نهایی رسید و مثلث‌های حذف شده به همراه مثلث نهایی مجموعه مثلث‌های نهایی را می‌سازند. (شکل ۵-۳)

این الگوریتم فرض می‌کند که رئوس از قبل مرتب شده هستند و مانند الگوریتم‌های قشر محدب و مقعر پردازشی روی رئوس انجام نمی‌دهد. در مرحله اول زاویه درونی هر راس محاسبه می‌شود. اگر یک راس زاویه زیر ۱۸۰ درجه داشت، آن یک زاویه محدب است و در غیر این صورت یک زاویه بازتابی است. این مرحله $O(n)$ زمان می‌برد.

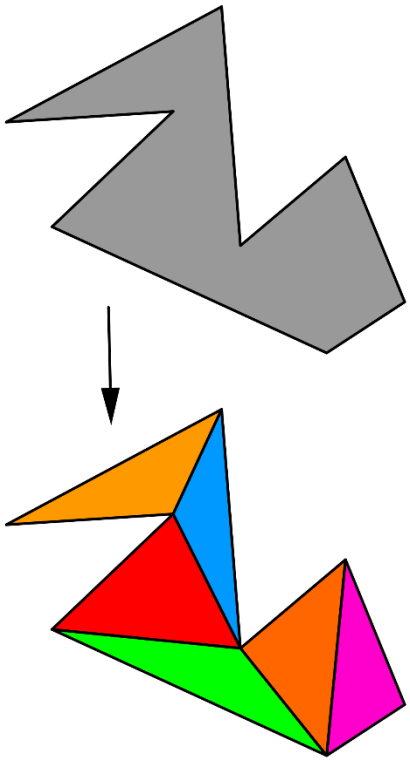
[1] Earclipping

در مرحله بعد تمام گوش‌ها پیدا می‌شود. یک گوشه علاوه بر اینکه زاویه محدب دارد، با ساخت مثلث با کمک راس قبل و بعد خود، موجب نمی‌شود که هیچ راس دیگری داخل مثلث ساخته شده بیفتد؛ در غیر این صورت آن راس به عنوان گوشه شناخته نمی‌شود.

در نهایت به ازای تمام گوشه‌ها از کوچکترین زاویه داخلی شروع می‌شود و مثلث گوشه حذف می‌شود. این عملیات تا وقتی ادامه پیدا می‌کند که $n - 2$ مثلث ساخته شده باشد. این مرحله $O(n^2)$ زمان می‌برد.

Algorithm 4: EAR CLIPPING ALGORITHM

```
    Input: A simple polygon P with n vertices V(v0,v1,...,vn-1)
    Output: A triangulation T with n–2 triangles
1   Compute interior angles of each vertex in P.
2   Indentify each vertex whether it is an ear tip or not.
3   while number of triangles in T < n–2 do
4       Find the ear tip vi which has the smallest interior angle.
5       Construct a triangle △(vi-1, vi, vi+1) and add it onto T.
6       Let vi be no longer an ear tip.
7       Update connection relationship of vi-1 and vi, vi and vi+1, vi-1 and vi+1.
8       Compute the interior angles of vi-1 and vi+1.
9       Refresh the ear tip status of vi-1 and vi+1.
10  end while
```



شکل ۵-۳ اجرای الگوریتم گوش‌بری روی یک چندضلعی مقعر

۵-۶-۷-الگوریتم های دیگر

الگوریتم‌های زیادی پس از الگوریتم گوش بُری معرفی شدند. در ابتدا این الگوریتم با پیچیدگی O(nlogn) حل شد [۳۸] و پس از بهبود هایی، نهایتا توسط رابرت تارجان[1] و کریستوفر ون ویک[2] به پیچیدگی O(nlog(logn)) رسید [۴۱]. روش‌های دیگری برای بهبود این روش مطرح شد و به پیچیدگی O(nlog*n) دست یافته شد که بسیار نزدیک به پیچیدگی خطی است [۴۲, ۴۳, ۴۴]. در نهایت در سال ۱۹۹۱ برنارد چزل[3]، با یک الگوریتم بسیار پیچیده، مسئله مثلثی سازی را در زمان خطی حل نمود [۴۵].

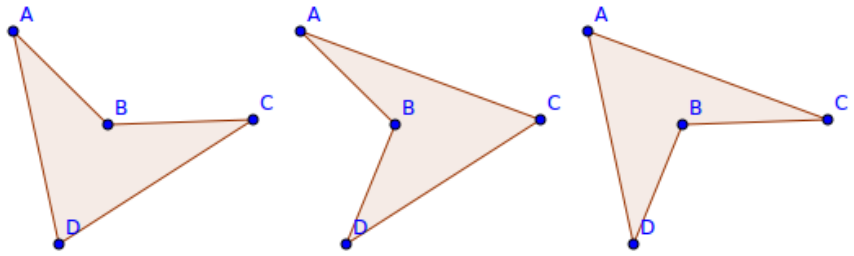
[1] Robert Tarjan
[2] Christopher Van Wyk
[3] Bernard Chazelle

در مقالـه‌ی یـک الگـوریتم O(nlog(logn)) بـرای مثلثـی‌سـازی چندضـلعی‌هـای سـاده، رابـرت تارجـان و همکاران ادعا کرده‌اند کـه مسـئله‌ی مثلثـی‌سـازی بسـیار سـاده‌تـر از مرتـب‌سـازی اسـت و در زمـان کمتـری از مرتب‌سازی انجام می‌شود. این بـا فـرض ایـن اسـت کـه مرتـب‌سـازی قابـل انجـام باشـد کـه همـانطور کـه ذکر خواهد شد، این کار بـرای چندضـلعی‌هـای مقعـر قابـل انجـام نیسـت و بایـد بـا توجـه بـه زمینـه ترتیـب رئوس بدست آورده شود.

۵-۶-۸-پیشنهاد الگوریتم بدست آوردن ترتیب رئوس

همانطور کـه گفتـه شـد پیچیـدگی زمـانی الگـوریتم‌هـای مثلثـی سـازی در بهتـرین حالـت خطـی اسـت امـا این بـه شـرطی اسـت کـه ترتیـب رئـوس را داشـته باشـیم. بـا داشـتن ترتیـب رئـوس بـرای چندضـلعی‌هـای محدب نیاز به اجرای الگوریتم قشر محـدب نیسـت و بـرای چندضـلعی‌هـای مقعـر بـا ایـن الگـوریتم مـی‌تـوان به ورودی رسید، اما سوال اینجاست بدست آوردن ترتیب رئوس چگونه انجام می‌شود.

در ابتـدای بخـش قبـل توضـیح داده شـد کـه در ایـن مسـئله هـیچ راسـی بیـرون ریختـه نمـی‌شـود و تمـام راس‌ها محیط قشر ما را تشکیل می‌دهنـد. بـا توجـه بـه اینکـه تـوری ورودی بـه هـر شـکلی مـی‌توانـد باشـد، فرض اینکه ترتیب نقاط تقاطع مثلث‌هـا بـا صـفحه بصـورت مرتـب شـده اسـت (سـاعتگرد یـا بـرعکس) غلـط است.



شکل ۵-۴ چند ضلعی مقعر با نقاط یکسان و قشر مقعر متفاوت

به علاوه با داشتن مجموعه‌ای از نقـاط، بدسـت آوردن قشـر مقعـر آنهـا ممکـن نیسـت؛ چـرا کـه ایـن مسـئله می‌تواند جواب‌های زیـاد و البتـه متفـاوتی داشـته باشـد (شـکل ۵-۴). الگـوریتم‌هـایی نیـز کـه مسـائلی مثـل فروشـنده دوره‌گـرد را حـل مـی‌کننـد قابـل اسـتفاده نیسـتند، چـون امکـان دارد اضـلاع چندضـلعی مـا، کوتاهترین اضلاع ممکن نباشند.

بـرای حـل ایـن مشـکل بایـد بـار محاسـباتی را در قسـمت محاسـبات تقـاطع‌هـا اضـافه کـرد کـه مـی‌توانـد سنگین باشد. در ادامه پیشنهاداتی برای حل این مسئله مطرح می‌شود.

در ابتدا باید مطمئن شویم هنگامی که یک مثلث با صفحه برنده تقاطع دارد، اضافه شدن نقاط تقاطع به لیست تقاطع‌ها طبق استاندارد انجام شود. این استاندارد به این شکل است که همواره جهت تقاطع‌های اضافه شده با جهت مثلث اصلی متناسب است.

فرض کنید راس a در بالای صفحه برنده قرار گرفته و b و c در زیر آن قرار دارند. جهت رئوس هم به شکل abc است. اگر ab تقاطع i و ac تقاطع i' را بسازد در اینجا ابتدا جهت جدید ii' است. لازم می‌بینم یادآوری کنم که این جهت اختیاری است، ولی باید در طول اجرای الگوریتم متداوم باشد.

در حالت دیگر اگر a تنها باشد ولی در پایین صفحه برنده واقع باشد و ab تقاطع i و ac تقاطع i' را بسازد، این بار جهت جدید i'i خواهد بود.

نکته‌ی دیگر حالت‌هایی است که فقط یک تقاطع داریم. در این حالت کافی است نقطه‌ای که روی صفحه قرار دارد را نیز یک نقطه‌ی تقاطع جدید در نظر بگیریم و از قرارداد تعیین شده تخطی نکنیم. با این روش می‌توانیم این حالات را نیز در این روش مدنظر قرار دهیم.

پس از اینکه به قرارداد منسجمی برای ترتیب اضافه شدن تقاطع‌ها رسیدیم باید راهی پیدا کنیم تا آنها را به هم متصل کنیم و یک زنجیره کامل بسازیم. حال که مجموعه‌ای از جفت رئوس جدید داریم می‌توانیم از آنها برای اتصال هم استفاده کنیم. فرض کنید هر جفت یک دوقطبی است که هر قطب آن فقط به قطب مشابه می‌چسبد. در اینجا منظور از قطب همان رئوس می‌باشند.

با توجه به توپولوژی مثلثی می‌دانیم هر ضلع به ضلع دیگری متصل است، پس می‌توان ادعا کرد هر تقاطع که حاصل برخورد یک ضلع به صفحه برنده است دو بار تکرار می‌شود، با فرق اینکه یکبار در ابتدای جفت ساخته شده قرار می‌گیرد و یکبار در انتهای آن.

فرض کنید دو راس v_0 و v_1 اعضای یک جفت راس باشند. در اینجا ما باید به دنبال جفتی بین جفت‌های اضافه شده باشیم که v_1 را به عنوان عضو اول خود داشته باشد (برای مثال v_1 و v_2). با پیدا کردن چنین جفتی می‌توان ادعا کرد v_0 و v_1 و v_2 باید پشت سر هم قرار گیرند.

با ادامه دادن این عملیات به حلقه‌ی نهایی می‌رسیم و وقتی حلقه به پایان می‌رسد که تمام مثلث-های پیمایش شده باشند. در نهایت به یک زنجیره می‌رسیم که برای چندضلعی‌های محدب ورودی مثلثی سازی پنکه‌ای است و برای چندضلعی‌های محدب ورودی الگوریتم گوش بُری است.

نکته‌ی دیگری که باید مدیریت شود، ابتدای زنجیره است. فرض کنید با داشتن زنجیره‌ی v_0 و v_1، یک جفت جدید v_5 و v_0 اضافه شود. در این حالت باید v_5 به ابتدای لیست اضافه شود.

این روش پیشنهادی مانند بسیاری از الگوریتم‌های دیگر قابلیت مدیریت توری‌های غیرساده را ندارد. یکی دیگر از نقاط ضعف این است که موازی پذیری زیادی ندارد؛ این به علت آن است که پردازش صحیح زنجیره‌های ساخته شده نیازمند آن است که زنجیره دست نخورد. شدت این مشکل با بکارگیری قفل‌ها برای شبیه‌سازی عملیات اتمی در دستکاری زنجیره، تقلیل می‌یابد.

به علاوه، می‌بایست خاطرنشان کرد که این روش برای ساخت ورودی برای الگوریتم‌های مثلثی‌سازی طراحی شده است. همواره در این مسائل، ترتیب درست رئوس به عنوان ورودی داده می‌شود و این الگوریتم سعی در بدست آوردن این ترتیب دارد.

Algorithm 5: VERTEX ORIENTATION ALGORITHM

```
Input: A list P of vertex pairs in right winding direction
Output: A list C containing the vertices in right orientation

Chains := a list of vertex groups with their heads and tails
Add P[0] to first vertex group of Chains with it's first element set as head and
the second set as the tail of it.

for i = 1, ..., length of P do
    for j = 0, ..., length of Chains do
        if first element of P[i] is equal to tail of Chains[j] then
            Add second element of P[i] to vertex group of Chains[j]
            Update tail of Chains[j] to second element of P[i]
            if there is an element k in Chains with it's head equal to second
element of P[i]
                Remove first element of vertex group of k
                Append the vertex group of k to vertex group of Chains[j]
                Update tail of Chains[j] to tail of k
                Remove k from Chains
        else if second element of P[i] is equal to head of Chains[j] then
            Add first element of P[i] to first element of vertex group of Chains[j]
            Update head of Chains[j] to first element of P[i]
        else
            Add P[i] to Chains with it's first element set as head and the second
set as the tail of it.
return P[0] vertex group
```

برای بهبود پیچیدگی این الگوریتم می‌توان از دو نگاشت هش استفاده کرد که یکی از آنها با کلید head و دیگری با کلید tail کار می‌کند و هر دو از رفرنس مشترک برای نگهداری زنجیره‌ها نگهداری می‌کنند. با این کار می‌توان به پیچیدگی خطی رسید.

۵-۷-تو خالی کردن توری مورد برش

برای توخالی کردن همانطور که در فصل چهار توضیح داده شد، نیاز به تشکیل مثلثات جدید داریم. برای این کار کافی است برای هر کدامیک از توری‌ها، عملیات افزودن مثلث‌ها را دوباره انجام دهیم. این‌بار تفاوت در ترتیب افزودن مثلث‌ها است.

برای اینکه پشت یک جسم سه‌بعدی دیده شود، باید هندسه‌ی پشت آن بصورت پادساعتگرد به داده توری اضافه شود. یکی از نقاط ضعف این روش تعداد راس‌های دوبرابر است که با توجه به اینکه این رئوس بردار عمود متفاوتی برای نورپردازی طبیعی خواهند داشت، استفاده مجدد از رئوس تکراری امکان پذیر نیست.

برای محاسبه‌ی بردار عمود رئوس درونی، کافی است بردار عمود راس متناظر را برگردانیم. این کار به سادگی و با قرینه کردن بردار عمود صورت می‌گیرد.

در اینجا لازم است روش دیگری که برای توخالی سازی مورد آزمایش قرار گرفت و هزینه‌ی محاسباتی کمتری داشت را معرفی کنم. این روش به اینگونه است که با تغییر سایه‌زن شی، کاری کنیم که درون آن نیز قابل دیدن باشد و در واقع عملیات جمع‌آوری سطوح پشتی اتفاق نیفتد. این روش آزمایش شد ولی در عمل جسم توخالی سایه‌پردازی غیرطبیعی داشت و به همین دلیل از این روش دست کشیده شد.

۵-۸-برش یک شی

همانطور که در فصل چهار گفته شد، اشیای سه‌بعدی دارای ویژگی مکان، چرخش و اندازه هستند که این اطلاعات در داده توری ذخیره نمی‌شود. با توجه به اینکه الگوریتم برش طراحی شده فقط با داده توری کار می‌کند، لازم است قبل از شروع برش محاسباتی انجام شود؛ با این کار اطلاعات صفحه برنده و شی مورد برش از مبدا محاسبه می‌شود که دقیقا همخوان به اطلاعات توری باشد که شی را در نقطه مبدا، با چرخش صفر و اندازه یک فرض کرده است.

برای این‌کار نیاز به محاسباتی است که نیازمند ماتریس چهار در چهار اطلاعات شی مورد برش است. این ماتریس بیانگر اندازه، چرخش و مکان یک شی در سه بعد است. با ضرب این ماتریس در بردار چهار بعدی (x,y,z,1) می‌توان برایند تمام عملیات‌های تغییر اندازه، چرخش و تغییر مکان هر نقطه را بدست آورد.

برای تغییر اندازه یک شی می‌توان از ماتریس زیر استفاده کرد [۴۶] .

$$\begin{bmatrix} S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S_1.x \\ S_2.y \\ S_3.z \\ 1 \end{pmatrix}$$

معادله ۵–۱۵

به همین شکل برای چرخش روی محور x می‌توان از ماتریس معادله‌ی ۵-۱۶ کمک گرفت [۴۶].

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ cos\theta.y - sin\theta.z \\ sin\theta.y + cos\theta.z \\ 1 \end{pmatrix}$$

معادله ۵–۱۶

همینطور چرخش روی محور y و z در ماتریس معادله ۵-۱۷ و ۵-۱۸ آورده شده است [۴۶].

$$\begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sing\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

معادله ۵–۱۷

$$\begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

معادله ۵–۱۸

برای انتقال نیز از ماتریس معادله‌ی ۵-۱۹ کمک گرفته می‌شود. [۴۶].

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix} \qquad \text{معادله ۵-۱۹}$$

در نهایت با ضرب پیاپی این ماتریس‌ها می‌توان به هر نتیجه دلخواهی رسید. نکته در خور توجه این است که ترتیب ضرب ماتریس‌ها مهم هست و تعیین کننده این است که کدام عملیات‌ها زودتر انجام گیرند. با داشتن این ماتریس می‌توان محاسباتی را انجام داد که معادله‌ی صفحه را از یک معادله در مختصات جهان سه‌بعدی به مختصات محلی شی تبدیل می‌کند که در برش از آن استفاده می‌کنیم.

با فرض اینکه ترکیب ماتریس‌های معرفی شده را که نمایانگر موقعیت شی است T بنامیم، بردار عمود محلی از طریق معادله ۵-۲۰ بدست می‌آید.

$$P'_{normal} = \|(T^T)^{-1} \times P_{normal}\| \qquad \text{معادله ۵-۲۰}$$

برای محاسبه‌ی بردار مکان محلی نیز می‌توان از معادله ۵-۲۱ کمک گرفت.

$$P'_{position} = T^{-1} \times P_{position} \qquad \text{معادله ۵-۲۱}$$

بعد از محاسبات هم از مکان و چرخش و اندازه جسم اصلی استفاده می‌شود تا اجسام تشکیل شده از برش با همان اطلاعات در محیط ظاهر شوند.

فصل ۶
جزئیات پیاده‌سازی

جزئیات پیاده سازی

در این فصل به جزئیات پیاده‌سازی این چارچوب پرداخته می‌شود و نگاه عمیق‌تری به روش حل مسئله برش اشیاء در سه‌بعد خواهیم داشت. در اینجا لازم می‌بینم یادآوری کنم کدهای نشان داده شده در این فصل مستعد تغییر در نسخه‌ی نهایی هستند و تغییر ساختار کد بخاطر خوانایی بهتر در پایان‌نامه غیر قابل امتناع می‌باشد.

شکل ۶-۱ خط لوله اجرای این الگوریتم را نشان می‌دهد. (از راست به چپ)

شکل ۶-۱ خط لوله اجرای الگوریتم برش

در ابتدای این فصل یک تکنولوژی پردازش موازی را معرفی کرده و بررسی می‌کنیم که آیا این تکنولوژی برای کاربرد برش قابل استفاده است یا خیر. در ادامه وارد مبحث برش می‌شویم و نحوه انتزاع کلاس‌ها را معرفی می‌کنیم و پیاده‌سازی‌های برش، از جمله محاسبات قبل از انجام برش، برش، قشر محدب، مثلثی سازی و در نهایت ساخت توری را بررسی می‌کنیم. در نهایت وارد جزئیات عملیات پس از برش می‌شویم و به معرفی ابزارهای توسعه داده شده دیگر می‌پردازیم.

۶-۱-سایه‌زن محاسباتی [۴۷]

همانطور که در فصل چهار توضیح داده شد، سایه‌زن‌ها، برنامه‌های کامپیوتری هستند که توسط کارت گرافیکی اجرا می‌شوند و تصویر نهایی یک شی را تعیین می‌کنند. با توجه به قدرت زیاد کارت-های گرافیک امروزی استفاده از آنها صرفا برای مسایل گرافیکی، کم‌لطفی در حق آن‌هاست و باید از این ظرفیت حداکثر استفاده را نمود.

سایه‌زن‌های محاسباتی که در علوم کامپیوتر به هسته‌های محاسباتی شناخته می‌شود، برنامه‌های هستند که با هدف توان عملیاتی بالا طراحی شده اند. این برنامه‌ها از برنامه اصلی جدا هستند ولی توسط آن‌ها فراخوانی و استفاده می‌شوند. نکته‌ی دیگر این است که برخلاف سایه‌زن‌های معمولی، سایه‌زن‌های محاسباتی به سخت‌افزار محدودیت ندارند و نیاز به نوشتن چند سایه‌زن محاسباتی برای کارت گرافیک‌های مختلف نیست. البته این محدودیت نداشتن به دلیل تبدیل راحت این برنامه‌ها به معماری‌های واحد پردازنده گرافیکی است.

انجام محاسبات سایه‌زن‌های محاسباتی در خارج از خط لوله رندر صورت می‌گیرد. این به این دلیل است که کارت گرافیک در هنگام محاسبات واحد پردازنده مرکزی بیکار است و منتظر دریافت دستور برای رسم روی صفحه است که به آن فراخوانی رسم می‌گویند. در این بازه‌های زمانی می‌توان از قدرت کارت گرافیکی به عنوان محاسبه‌گر در کنار واحد پردازنده مرکزی استفاده کرد.

این برنامه‌ها را می‌توان در بسته‌های یک تا سه بعدی فراخوانی کرد و با توجه به واحدهای جریان چندهسته‌ای، هسته‌های محاسباتی در بلوک‌هایی فراخوانی می‌شوند و از قدرت پردازش چندهسته-ای بهره‌برداری می‌شود.

داده‌هایی که در اختیار این برنامه‌ها قرار داده می‌شود باید از طریق بافرهای محسباتی از واحد پردازنده مرکزی وارد واحد پردازنده گرافیکی شوند که این گذر بین دو سخت‌افزار برای دستگاه‌هایی که از پردازنده یکپارچه استفاده نمی‌کنند زمانبر است و باید بررسی شود آیا سود استفاده از کارت گرافیکی به احتمالا دو بار سربار گذر بین دو سخت‌افزار می‌چربد یا خیر.

در کاربردهایی از قبیل شبیه‌سازی کلان داده‌ها یا ساختن تصاویر کامپیوتری با بسیار ابعاد بالا، استفاده از سایه‌زن‌های گرافیکی تاثیر به‌سزایی در کارایی نرم‌افزار دارد. با توجه با اینکه توری‌ها در بازی‌های ویدئویی برای کارایی بهتر پخته می‌شوند تا رئوس کمتری داشته باشند، با پردازش آن‌ها توسط کارت گرافیکی، استفاده حداکثری از این دستگاه‌های پرقدرت صورت نمی‌گیرد.

در کاربرد برش توری، کاری که واحد پردازنده گرافیکی می‌تواند انجام دهد، محاسبه نقاط برخورد تمامی مثلث‌ها با صفحه است. علاوه بر این برگرداندن آن در قالب دو توری جدید نیز می‌تواند توسط واحد پردازنده گرافیکی بصورت موازی انجام شود ولی بخاطر پیچیدگی الگوریتم غیر نمایی و تعداد نه چندان زیاد مثلث‌ها برای کامپیوترهای امروزی استفاده از این روش پیشنهاد نمی‌شود؛ زیرا پاس دادن بافر محاسباتی این داده ها به کارت گرافیکی و بازگرداندن آن زمان بیشتری می‌برد.

یک پیشنهاد این است که عملیات برش کاملا در کارت گرافیکی و در خط لوله رندر انجام شود که این‌کار فقط در زمانی امکان پذیر است که زمان کافی برای کشیدن روی صفحه داشته باشیم تا نرخ فریم کم نشود و این در وقتی است که تصویر پیچیده‌ای برای رسم روی صفحه نداشته باشیم. در این روش نیازی به پاس دادن دوباره اطلاعات رئوس نداریم چون به هر حال این داده‌ها قبل از فراخوانی رسم به کارت گرافیک گذر می‌کنند.

۶-۲-بازنمایی رئوس

بصورت کلی هر راس در سه‌بعد حتما داده مکانی را دارد. برای بازی‌هایی که نورپردازی دارند (که تقریبا همه‌ی بازی‌ها از این ویژگی استفاده می‌کنند) داده‌ی بردار عمود نیز لازم است و اجسامی که بافت‌پذیر هستند، نیاز به داده یو وی دارند. این سه ویژگی رئوس تقریبا در همه توری‌هایی که در بازی‌ها به کار می‌رود، استفاده می‌شود.

یک راس برای وجود، کافی است مکان داشته باشد و ممکن است اطلاعات یو وی و عمود آن بعد از پردازش‌هایی اضافه شوند.

```
public class Vertex {
    Vector3 _position;
    Vector2 _uv;
    Vector3 _normal;
}
```

شکل ۶–۲ کد کلاس راس

۶-۳-بازنمایی مثلث

همانطور که دیده می‌شود، یک مثلث از سه راس تشکیل شده است.

```
public class Tri {
    private Vertex _vertA, _vertB, _vertC;

    public Intersection Split(Plane plane) {
        return Intersector.Intersect(plane, this);
    }
}
```

شکل ۶–۳ کد کلاس مثلث

۶-۴-بازنمایی صفحه

```
public class Plane {
    private Vector3 _position;
    private Vector3 _normal;

    public Plane(Vector3 position, Vector3 normal) {
        _position = position;
        _normal = normal;
    }

    public Plane(Vector3 a, Vector3 b, Vector3 c) {
        Normal = Vector3.Cross(b - a, c - a);
        _position = a;
    }

    public Vector3 Normal {
        get => _normal;
        private set => _normal = Vector3.Normalize(value);
    }

    public float Distance { get => Vector3.Dot(_position, _normal); }
    public Vector3 Position { get => _position; }

    public PointToPlaneRelation GetPointToPlaneRelation(Vector3 p) {}

    public TrianglePlaneRelation GetTriangleToPlaneRelation(Tri tri) {}
}
```

شکل ۶-۴ کد کلاس صفحه

همانطور که دیده می‌شود، هر صفحه دو متغیر مکان و بردار عمود دارد، برای راحتی، ویژگی فاصله که در فصل ۵ به آن پرداختیم علاوه بر بردار عمود از بیرون در دسترس است. تابع GetPointToPlaneRelation که با ورودی یک نقطه (بردار سه‌بعدی) یا یک مثلث، رابطه‌ی نقاط با صفحه را حساب می‌کند و تابع GetTriangleToPlaneRelation رابطه‌ی مثلث با صفحه را محاسبه می‌کند که درباره آن در بخش‌های بعدی بحث می‌شود.

۶-۵-محاسبه برخورد با صفحه

فرض کنید یک صفحه سه‌بعدی داریم که نمایانگر صفحه برنده است. صفحه‌ای که در شبیه‌سازی گرافیکی قرار گرفته است باید بتواند اشیایی که با آن برخورد داشته‌اند را تشخیص دهد. به علاوه به دلیل آنکه این صفحه در دنیای سه‌بعدی محدود است، نباید آن را سرسری تبدیل به صفحه نامحدود کنیم، چون اگر چیزی که کاربر می‌بیند با چیزی که باید اتفاق بیفتد، تجربه کاربری بدی به کاربر انتقال می‌یابد.

برای اینکه تشخیص بدهیم که آیا یک صفحه به اجسام دیگر برخورد داشته است یا نه، می‌توانیم از موتور فیزیکی برای محاسبات برخورد کمک بگیریم. با توجه به اینکه اشیای بازی در یونیتی از اجزایی تشکیل شده‌اند که رفتار آن شی را مشخص می‌کند باید علاوه بر شی مورد برش، به صفحه‌ی برنده نیز ویژگی برخورد پذیری را اضافه کنیم.

خوشبختانه یونیتی این قابلیت را به توسعه‌دهندگان می‌دهد که تاثیر برخورد فیزیکی را تنظیم کنند. در این مسئله میدانیم صفحه برنده یک صفحه مجازی است و نباید روی اشیای دیگر تاثیر فیزیکی داشته باشد؛ به طور دقیق تر یک صفحه باید بتواند از میان یک شی رد شود، بدون اینکه تاثیری روی آن داشته باشد.

در یونیتی با داشتن ویژگی برخوردپذیری و قطع کردن تاثیر فیزیکی برخورد با استفاده از می‌توانیم از موتور فیزیکی بهره ببریم تا تشخیص دهیم چه اشیایی با صفحه برخورد داشته اند. برای این کار کافی است تابع OverlapBox را فراخوانی کنیم تا لیست تمامی اشیایی که در جعبه‌ی ورودی هستند و برخوردکننده فیزیکی دارند را بازگرداند.

تابع OverlapBox که در کتابخانه Physics در یونیتی در دسترس است [۴۸] با گرفتن مرکز، اندازه و چرخش جعبه می‌تواند لیستی از تمامی اشیایی که دارای برخوردکننده فیزیکی هستند و با جعبه‌ی داده شده همپوشانی دارند را برگرداند.

علاوه بر این می‌توان به این تابع ورودی‌های دیگری از جمله ماسک لایه داد تا فقط اشیایی را برگرداند که در آن لایه خاص هستند. از آنجایی که ما فقط برخورد با اشیای برش‌پذیر را نیاز داریم می‌توانیم صرفا از این لایه استفاده کنیم.

برای بدست آوردن دیگر ورودی‌های این تابع کافی است از مکان و چرخش صفحه در دنیای سه-بعدی استفاده کنیم. با توجه به اینکه صفحه ضخامت ندارد، برای ابعاد طول و عرض آن، طول و عرض خودش را میگیریم و ضخامت آن را برابر مقدار کوچکی قرار می‌دهیم که جعبه‌ی مورد نظر ساخته شود. البته می‌توان برای سادگی از اندازه برخوردکننده این صفحه استفاده کرد که نیاز به کار اضافه نداشته باشیم.

```
_size = _plane.GetComponent<BoxCollider>().size;
Collider[] hitColliders = Physics.OverlapBox(_planeTransform.position,
                                             _size,
                                             _planeTransform.rotation,
                                             _layer);
```

شکل ۶–۵ تشخیص هم‌پوشانی مربعی

۶-۶-محاسبه جسم بین دو کلیک

از آنجایی که کلیک روی صفحه، ورودی دوبعدی است (با متغیر طول و عرض) نیاز به پردازش‌هایی داریم تا آن را تبدیل به داده سه‌بعدی بکنیم. برای این کار می‌توان از توابع دوربین موجود در یونیتی کمک گرفت. تابع ScreenToWorldPoint یک ورودی صفحه را تبدیل به نقطه‌ای در دنیای سه-بعدی می‌کند که در ادامه به آن خواهیم پرداخت.

نقاط موس، کلیک و لمس در موتور یونیتی با استفاده از ویژگی mousePosition در کلاس Input بدست می‌آید که جزئیات آن وابسته به سیستم‌عامل است. mousePosition یک بردار سه‌بعدی است که مقدار z آن صفر است. این عدد در واقع نشان‌دهنده این است که نقطه به دوربین چسبیده است و در عمق با آن فاصله‌ای ندارد. تغییر این عمق در تصویر نهایی تاثیری ندارد چرا که رابط کاربری به هر حال دوبعدی است. برای اطمینان از اینکه در تبدیل این نقطه به سه‌بعد، خطای ممیز شناور نداریم و عمق به درستی در خروجی تبدیل لحاظ شود، می‌توانیم عمق این نقطه را در فاصله کمی از دوربین قرار دهیم (مثلا "۰.۳").

```
lastMousePosition = Input.mousePosition;
lastMousePosition.z = Camera.main.nearClipPlane;
```

شکل ۶–۶ پردازش ورودی موس

در کد بالا بجای مقدار کم از صفحه‌ی شروع رندر دوربین استفاده کردیم که جایگزین مناسب و پویایی برای مقدار گفته شده است. nearClipPlane که صفحه‌ی نزدیک دوربین است تعیین می-کند اشیا از چه فاصله‌ای از دوربین نمایش داده شوند و از نمایش اشیای بسیار نزدیک به دوربین که باعث پوشش دید می‌شود، جلوگیری می‌کنند.

در ادامه با کد زیر می‌توانیم موقعیت سه‌بعدی نقطه را در فضای سراسری داشته باشیم.

```
worldPosition = Camera.main.ScreenToWorldPoint(lastMousePosition);
```

شکل ۶–۷ تبدیل ورودی موس به موقعیت سه‌بعدی

با داشتن دو نقطه کلیک در فضای سه‌بعدی، برای یافتن اشیای میان آنها می‌توانیم از تاباندن اشعه از سمت دوربین استفاده کنیم. این تکنولوژی فیزیکی اجازه می‌دهد برخورد اشعه با اشیا را تشخیص دهیم.

برای این کار که با تابع RayCast انجام می‌شود، نیاز به نقطه‌ی شروع تابش، جهت تابش و برد تابش داریم که تعیین کننده این است که هر اشعه باید تا چه طولی تابیده شود (شکل ۶-۸). با داشتن دو نقطه‌ی کلیک سه‌بعدی به نام‌های startWorldPosition و endWorldPosition و تفریق مکان دوربین از آنها می‌توان به دو اشعه رسید که سر و ته برد جستجوی ما را تعیین می‌کنند.

شکل ۶-۸ نحوه عملکرد تابنده اشعه

با داشتن این دو بردار جهت، می‌توانیم با انجام دادن درون‌یابی برداری بین این‌دو می‌توانیم به بردار-های جهت میان این دو برسیم و در نهایت مجموعه‌ای از اشعه برای تاباندن خواهیم داشت.

```
RaycastHit hit;
for (float t = 0; t < 1; t += 0.1f) {
    Vector3 direction = Vector3.Lerp(startWorldPosition - transform.position, finishWorldPosition
- transform.position, t);
    if (Physics.Raycast(transform.position, direction, out hit, 100, _layerMask))
        hits.Add(hit.collider.gameObject);
}
```

شکل ۶-۹ تشخیص اشیای بین دو کلیک

در نهایت اشیای برخورد کرده در یک مجموعه اضافه می‌شوند که مطمئن شویم هر جسم حداکثر یکبار تکرار شود.

در این روش صفحه‌ی برنده با کمک سه نقطه‌ی دوربین و مکان دو کلیک در سه‌بعد ساخته می‌شود.

۶-۷-برش

بعد از بدست آوردن شی مورد برش و صفحه‌ی برنده می‌توانیم عملیات برش را انجام دهیم. برای این‌کار از تابع برش کمک می‌گیریم منتها قبل از آن لازم است انواع نسبت‌های مثلث با صفحه و نقطه با صفحه مشخص شود.

```
public enum PointToPlaneRelation {
    TOP = 1, BOTTOM = -1, SURFACE = 0
}

public enum TrianglePlaneRelation {
    NO_INTERSECTION, TWO_TRI, THREE_TRI
}
```

شکل ۶–۱۰ نسبت مثلث و نقطه به صفحه

برای رابطه‌ی یک مثلث با یک صفحه، طبقه‌بندی بر این اساس انجام شده است که برخورد یک صفحه با مثلث موردنظر چند مثلث جدید می‌سازد که در فصل پنج به آن پرداخته شد.

با این تعاریف می‌توانیم متوجه شویم که یک نقطه با یک صفحه چه رابطه‌ای دارد. این کار بصورت زیر انجام می‌گیرد.

```
public PointToPlaneRelation GetPointToPlaneRelation(Vector3 p) {
    float pointDistance = Vector3.Dot(p, _normal);

    float difference = pointDistance - Distance;

    if (difference > Constants.EPSILON) return PointToPlaneRelation.TOP;
    else if (difference < -Constants.EPSILON) return PointToPlaneRelation.BOTTOM;
    else return PointToPlaneRelation.SURFACE;
}
```

شکل ۶–۱۱ محاسبه نسبت نقطه به صفحه

در کد بالا Distance از ضرب داخلی نقطه و عمود صفحه بدست می‌آید که جزو ویژگی‌های صفحه محسوب می‌شود. برای اطمینان از اینکه خطای ممیز شناور نداریم از Constants.EPSILON استفاده می‌کنیم که برابر float.epsilon است.

برای اینکه نسبت یک مثلث با صفحه را حساب کنیم، می‌توانیم از روش زیر استفاده کنیم:

```
public TrianglePlaneRelation GetTriangleToPlaneRelation(Tri tri) {
    PointToPlaneRelation[] relations = new PointToPlaneRelation[3];

    var (a, b, c) = tri.GetPositions();
    var (ra, rb, rc) = (GetPointToPlaneRelation(a), GetPointToPlaneRelation(b),
GetPointToPlaneRelation(c));

    switch (Mathf.Abs((int)ra + (int)rb + (int)rc)) {
        case 3: return TrianglePlaneRelation.NO_INTERSECTION;
        case 2: return TrianglePlaneRelation.NO_INTERSECTION;
        case 0: return TrianglePlaneRelation.TWO_TRI;
        case 1:
            if (ra == PointToPlaneRelation.SURFACE || rb == PointToPlaneRelation.SURFACE || rc ==
PointToPlaneRelation.SURFACE)
                return TrianglePlaneRelation.NO_INTERSECTION;
            return TrianglePlaneRelation.THREE_TRI;
    }
}
```

شکل ۶–۱۲ محاسبه نسبت مثلث به صفحه

هدف از عددگذاری enum های رابطه نقطه و صفحه استفاده در اینجا بود. اگر نقطه‌ای بالای صفحه باشد مقدار آن ۱ است و اگر زیر صفحه باشد مقدار آن ۱- است؛ در غیر این صورت مقدار صفر می‌گیرد. اندازه‌ی جمع این مقادیر برای رئوس یک مثلث می‌تواند حالت کلی رابطه مثلث با صفحه را به ما بدهد.

اگر اندازه جمع ۳ باشد، یعنی یا هر سه راس ۱- بودند یا ۱، پس هیچ تقاطعی صورت نگرفته و هیچ مثلث جدیدی ایجاد نمی‌شود.

اگر اندازه جمع ۲ باشد یعنی دو راس در یک سمت و یک راس روی سطح قرار دارد و مانند حالت قبلی است.

به همین شکل برای ۰ و ۱ می‌توان نتیجه‌گیری کرد. فقط باید دقت کرد که جمع ۱، می‌تواند ناشی از دو حالت باشد که باید در کد مدیریت شود.

در ادامه به دلیل طولانی بودن حالت‌ها، یک نمونه محاسبه برخورد از هر حالت رابطه مثلث با صفحه نمایش داده می‌شود.

```
public static Intersection Intersect(Plane plane, Tri tri) {
    Intersection result = new Intersection();
    var (a, b, c) = tri.GetVertecies();
    var (ra, rb, rc) = plane.GetPointToPlaneRelations(tri);

    TrianglePlaneRelation triangleStatus = plane.GetTriangleToPlaneRelation(tri);

    if (triangleStatus == TrianglePlaneRelation.NO_INTERSECTION) return null;
    if (triangleStatus == TrianglePlaneRelation.TWO_TRI) {
        if (ra == PointToPlaneRelation.SURFACE) {
            Vertex i = Intersect(plane, b, c);
            result.AddNewVertex(new Vertex(i.Position, MathHelper.Project(i.Position,
plane.Normal, Vector2.one * .5f) ,plane.Normal));

            Tri tb = rb == PointToPlaneRelation.TOP ? new Tri(a, b, i) : new Tri(a, i, b);
            Tri tc = rc == PointToPlaneRelation.TOP ? new Tri(a, c, i) : new Tri(a, i, c);

            if (rb == PointToPlaneRelation.TOP) {
                result.AddTopTri(tb);
                result.AddBottomTri(tc);
            } else {
                result.AddTopTri(tc);
                result.AddBottomTri(tb);
            }
            return result;
        }
        else if (rb == PointToPlaneRelation.SURFACE) {}
        else {}
    }
    if (triangleStatus == TrianglePlaneRelation.THREE_TRI) {
        if (ra != rb) {
            Vertex i1 = Intersect(plane, a, b);
            result.AddNewVertex(new Vertex(i1.Position, MathHelper.Project(i1.Position,
plane.Normal, Vector2.one * .5f) ,plane.Normal));

            if (rc == ra) {
                Vertex i2 = Intersect(plane, b, c);
                result.AddNewVertex(new Vertex(i2.Position, MathHelper.Project(i2.Position,
plane.Normal, Vector2.one * .5f) ,plane.Normal));

                Tri ta = new Tri(a, i1, i2);
                Tri tc = new Tri(c, a, i2);
                Tri tb = new Tri(b, i2, i1);
```

```
                if (ra == PointToPlaneRelation.TOP) {
                    result.AddTopTri(ta);
                    result.AddTopTri(tc);
                    result.AddBottomTri(tb);
                } else {}
                return result;
            }
            else {}
        }
        else {}
    }
    return null;
}
```

شکل ۶–۱۳ ساخت مثلث های جدید از برخورد مثلث با صفحه

در اینجا از تابع Project استفاده شده که لازم است توضیح داده شود. همانطور که در فصل چهار گفته شد، موقعیت‌های توری در فضای محلی هستند و به همین‌خاطر خودشان نرمال شده اند. یک راس جدید که قرار است در پوشاندن سطح خالی توری جدید استفاده شود، نیاز به یو وی و بردار عمود متفاوتی دارد. برای بردار عمود کافی است از بردار عمود صفحه استفاده کنیم، زیرا در همان جهت است، منتها برای یو وی نیازمند آن هستیم که نقطه‌ی جدید در صفحه‌ی دوبعدی تصویر شود. برای این کار از تکنیک کاهش بعد با کمک بردار عمود استفاده می‌کنیم که در فصل پنج توضیح داده شد.

پیاده‌سازی تابع Intersect که یک صفحه و دو راس می‌گیرد به شکل زیر است که فرمول‌های آن در فصل پنج تشریح شدند.

```
public static Vertex Intersect(Plane plane, Vertex a, Vertex b) {
    Vector3 pNormal = plane.Normal;
    float pDistance = plane.Distance;
    Vector3 line = b.Position - a.Position;

    float t = (pDistance - Vector3.Dot(pNormal, a.Position)) / Vector3.Dot(pNormal, line);
    Vector3 normal = Vector3.Normalize(a.Normal * (1 - t) + b.Normal * t);
    Vector2 uv = a.UV * (1 - t) + b.UV * t;

    if (t >= -Constants.EPSILON && t <= 1 + Constants.EPSILON) return new
Vertex(Vector3.Lerp(a.Position, b.Position, t), uv, normal);
    else throw new System.InvalidOperationException("The given line and plane does not have an
intersection");
}
```

شکل ۶–۱۴ محاسبه نقطه تقاطع صفحه و خط

۶-۸-نگاشت و قشر محدب

برای بدست آوردن قشر محدب، نیاز به این داریم که بتوانیم تشخیص دهیم مساحت یک مثلث مقدار مثبت دارد یا منفی، به عبارت دیگر آیا راس سوم نسبت به ضلع متشکل از دو ضلع اول، به سمت ساعتگرد چرخیده یا پادساعتگرد که در معادله ۵-۱۴ محاسبه شد.

برای الگوریتم قشر محدل همانطور که در فصل قبل گفته شد، نیاز به مرتب سازی داریم. در اینجا از الگوریتم اسکن گراهام استفاده می‌کنیم که مرتب‌سازی در آن بر اساس زاویه ساخته شده بین پایین‌ترین نقطه با هر راس است. اگر دو راس زاویه‌ی یکسانی داشتند، هر کدام که فاصله‌ی نزدیک-تری داشت در مرتب‌سازی اولویت دارد. تکه کد زیر تابع مقایسه و محاسبه علامت مساحت مثلث را نشان می‌دهد.

```
public static int AreaSign(Vector2 a, Vector2 b, Vector2 c) {
    float area = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);

    if (area < 0) return -1; // Clockwise
    if (area > 0) return 1; // Counter-clockwise
    return 0; // 3 vectors are colinear
}

public static int Compare(Vector2 p1, Vector2 p2) {
    int areaSign = AreaSign(referencePoint, p1, p2);

    if (areaSign == 0) {
      if (Vector2.Distance(referencePoint, p2) > Vector2.Distance(referencePoint, p1)) return -1;
      else return 1;
    } else {
      if (areaSign == 1) return -1;
      else return 1;
    }
}
```

شکل ۶–۱۵ محاسبه علامت مساعت مثلث و مرتب سازی رئوس

در کد صفحه بعد هم تابع محاسبه‌ی قشر محدب دیده می‌شود. برای این مسئله صرفا مرتب سازی کافی است و نیاز به پردازش بیشتری نداریم ولی این کار باعث توپولوژی تمیزتری می‌شود و تعداد کمتری از رئوس را انتخاب می‌کند. به طور دقیق تر اگر چند راس در امتداد هم باشند، فقط یکی از آنها انتخاب می‌شود. علاوه بر این گلوگاه این الگوریتم قسمت مرتب سازی است و حلقه‌ی while در O(n) اجرا می‌شود.

```
public static int[] CalculateConvexHull(Vertex[] vertices, Vector3 normal) {
    maps = MathHelper.Project(vertices, normal);
    Stack<Map2D> hulls = new Stack<Map2D>();

    int lowestYIndex = 0;
    float lowestYValue = maps[0].position.y;

    for(int i = 0; i < maps.Length; i++) {
        if (maps[i].position.y <= lowestYValue) {
            lowestYIndex = i;
            lowestYValue = maps[i].position.y;
        }
    }

    referencePoint = maps[lowestYIndex].position;


    Array.Sort(maps, (x, y) => Compare(x.position, y.position));

    hulls.Push(maps[0]);
    hulls.Push(maps[1]);

    for (int i = 2; i < maps.Length; i++) {
        var p = hulls.Pop();
        while (hulls.Count != 0 &&
               AreaSign(hulls.Peek().position, p.position, maps[i].position) <= 0) {
            p = hulls.Pop();
        }
        hulls.Push(p);
        hulls.Push(maps[i]);
    }


    int[] convexHullIndices = new int[hulls.Count];

    int index = 0;
    while(hulls.Count != 0) convexHullIndices[index++] = hulls.Pop().index;

    return convexHullIndices;
}
```

شکل ۶–۱۶ پیاده‌سازی الگوریتم اسکن گراهام

در کد صفحه قبل Map2D نقطه‌ی تصویر شده در دو بعد و اندیس نقطه‌ی اصلی در لیست ورودی را نگه‌داری می‌کند. در نهایت این اندیس‌ها پس داده می‌شود. دلیل کاهش بعد در اینجا این است که الگوریتم مورد استفاده ابعاد بالاتر را پشتیبانی نمی‌کند و در هر حال نقاط چندضلعی از قبل همگی در یک صفحه قرار دارند.

۶-۹-مثلثی سازی

برای پر کردن قسمت خالی ایجاد شده پس از برش، نیاز داریم مثلثی سازی انجام دهیم. ورودی این الگوریتم راس‌هایی هستند که محیط چندضلعی جدید را می‌سازند و خروجی این الگوریتم یک لیست از مثلث‌های جدید تشکیل شده است. علاوه بر این برای اینکه بردار عمود مثلث‌های جدید را داشته باشیم، به عمود صفحه‌ی برنده نیاز داریم.

تابع زیر مثلثی سازی را انجام می‌دهد. در ابتدا نیاز به محاسبه‌ی نقطه مرکزی داریم که مثلثی‌سازی با توپولوژی پنکه‌ای مرکزی انجام شود. سپس با کمک الگوریتم قشر محدب، اندیس رئوس را به ترتیب می‌گیریم و در نهایت مثلث‌ها را میسازیم.

```
static List<Tri> Triangulate(Vertex[] newVertices, Vector3 normal) {
    List<Tri> fillTriangles = new List<Tri>();

    Vertex fanMiddleVertex = MathHelper.Average(newVertices);
    fanMiddleVertex.Normal = normal;
    fanMiddleVertex.UV = MathHelper.Project(fanMiddleVertex.Position, normal, Vector2.one * .5f);


    int[] hullIndices = ConvexHull.CalculateConvexHull(newVertices, normal);

    for (int i = 0; i < hullIndices.Length - 1; i++) {
        fillTriangles.Add(new Tri(
            fanMiddleVertex,
            newVertices[hullIndices[i]],
            newVertices[hullIndices[i + 1]]
            ));
    }

    fillTriangles.Add(new Tri(
            fanMiddleVertex,
            newVertices[hullIndices[hullIndices.Length - 1]],
            newVertices[hullIndices[0]]
            ));

    return fillTriangles;

}
```

شکل ۶–۱۷ مثلثی سازی

۶-۱۰-ساخت توری از لیست مثلثات

برای این‌کار کافی است آرایه‌های مورد نیاز از مکان، عمود و یو وی رئوس بسازیم و در آرایه‌ی اندیس‌ها آن‌ها را اضافه کنیم. سپس با ساختن یک توری خالی و اضافه کردن این داده‌ها در آن، توری از مثلث‌ها ساخته می‌شود و آماده استفاده در بازی می‌شود.

```
public static Mesh CreateMeshFromTriangles(List<Tri> triangles) {
    Mesh result = new Mesh();
    int vertexCount = triangles.Count * 3;

    Vector3[] vertecies = new Vector3[vertexCount];
    Vector3[] normals = new Vector3[vertexCount];
    Vector2[] uvs = new Vector2[vertexCount];
    int[] indices = new int[vertexCount];

    for(int i = 0; i < triangles.Count * 3; i += 3) {
        (vertecies[i], vertecies[i + 1], vertecies[i + 2]) = triangles[i / 3].GetPositions();
        (normals[i], normals[i + 1], normals[i + 2]) = triangles[i / 3].GetNormals();
        (uvs[i], uvs[i + 1], uvs[i + 2]) = triangles[i / 3].GetUVs();


        indices[i] = i;
        indices[i + 1] = i + 1;
        indices[i + 2] = i + 2;
    }

    result.vertices = vertecies;
    result.normals = normals;
    result.uv = uvs;
    result.SetTriangles(indices, 0, false);

    return result;
}
```

شکل ۶–۱۸ ساخت توری از مثلثات

یونیتی اجازه می‌دهد توری ساخته شده در زمان اجرا بهینه‌سازی شود. با بهینه‌سازی توری، ترتیب رئوس طوری مرتب می‌شود که کارایی رسم آن روی صفحه نمایش افزایش یابد. البته این عملیات برای توری‌های پیچیده زمانبر است و پیشنهاد نمی‌شود.

۶-۱۱-موتور تویین

یکی دیگر از ابزارهایی که در طول توسعه‌ی این پروژه طراحی شد، موتورِ بِینی است که در صنعت با نام تویین یاد می‌شود. این موتور می‌تواند انیمیشن‌های پویا با ورودی متغیر طراحی کند که برای نمایش جدا شدن دو تکه‌ی برش خورده شده از هم استفاده می‌شود.

نحوه‌ی کلی کار این موتور به این گونه است که در هر فریم، به میزان فاصله‌ی زمانی طی شده از فریم قبلی حرکت می‌کند تا وقتی که به هدف انیمیشن برسد. این حرکت می‌کند بصورت خطی باشد و یا با کمک توابعی بصورت نرم‌تر باشد. یک نمونه از این توابع که بدون نرم‌سازی است در زیر آورده شده است.

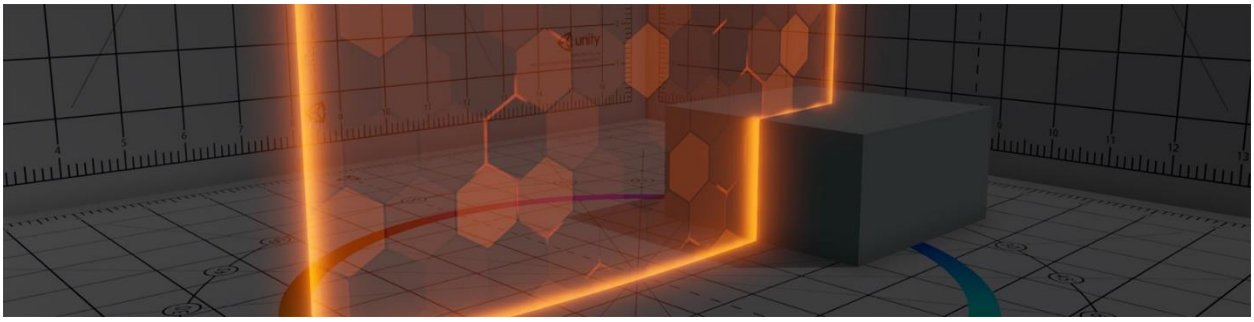
```
IEnumerator MoveCoroutine(Vector3 position, Vector3 direction, float duration, float amount) {
    direction = direction.normalized;
    Vector3 originalPosition = transform.position;
    float t = 0;
    while (t <= duration) {
        transform.position = originalPosition + direction * Mathf.Lerp(0, amount, t / duration);
        t += Time.deltaTime;
        yield return null;
    }
}
```

شکل ۶–۱۹ نمونه‌ای از روتین پویانمایی با استفاده از کد

کد بالا در روتین‌هایی اجرا می‌شود که برخلاف توابع عادی در یونیتی، می‌تواند بیشتر از یک فریم عمر کند و با قدرت آن‌ها می‌توان به انیمیشن‌های متغیر دست یافت.

۶-۱۲-شیدر محاسبات برخورد [۴۹]

همچنین برای نمایش دادن مکان برخورد صفحه با اشیا یک سایه‌زن در این پروژه پیاده‌سازی شده که می‌تواند با کمک متغیر عمق دوربین به چنین اثری دست یابد (شکل ۶-۲۰). همانطور که در فصل دو گفته شد برای این‌کار از زبان اچ ال اس ال استفاده است ولی با توجه به پیچیدگی زبان‌های سایه‌زنی و این حقیقت که این سایه‌زن مربوط به عملیات برش نیست از طرح جزئیات بیشتر صرفنظر می‌شود.

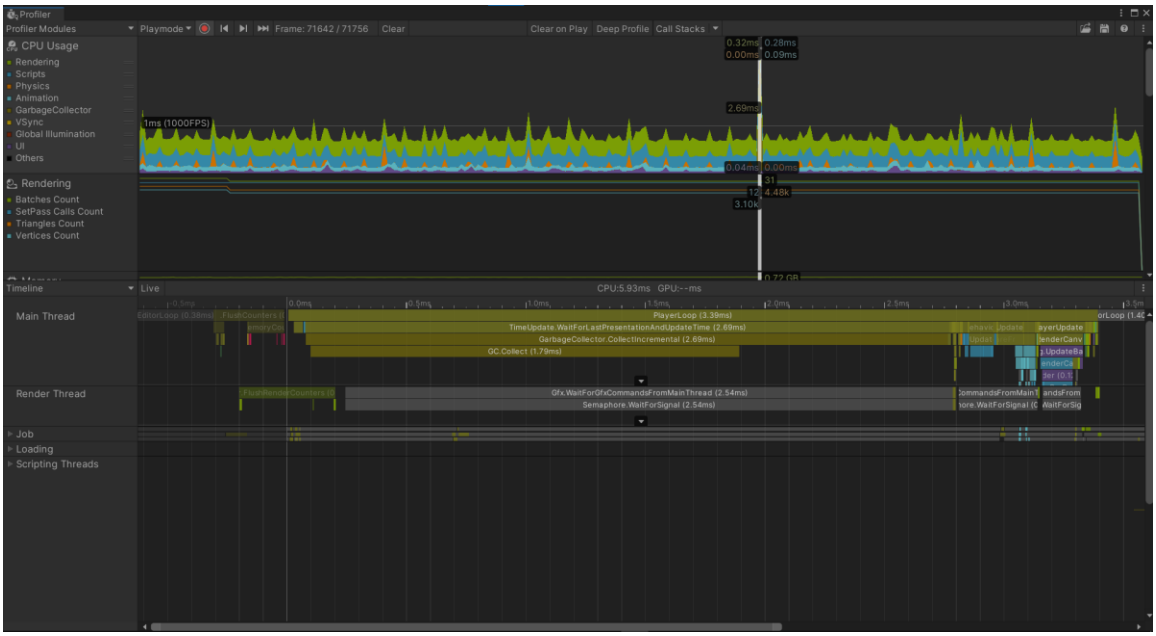


شکل ۶–۲۰ نمونه ای از سایه‌زن تشخیص برخورد با اشیا [۴۹]

فصل ۷
اندازه‌گیری و آزمایش

اندازه گیری و آزمایش

در این فصل به آزمایش مختصر از اجرای الگوریتم می‌پردازیم. همانطور که در فصل اول گفته شد، یکی از چالش‌های این مسئله بودجه زمانی آن است و زمان اجرای الگوریتم برش نباید بیش از حد زیاد باشد. برای اندازه‌گیری‌های این فصل از ابزار نمایه‌ساز[1] یونیتی استفاده می‌کنیم. (شکل ۷-۱)



شکل ۷-۱ رابط کاربری نمایه‌ساز یونیتی

برای اندازه‌گیری ابتدا از برش اشیایی که در تعریف پروژه ذکر شد استفاده می‌کنیم. این اشیا عبارت‌اند از صفحه، مکعب، کره و استوانه. سپس یک دایره پیچیده تر با تعداد رئوس بسیار بالاتر را مورد بررسی قرار می‌دهیم. باید توجه نمود که ابزار نمایه‌ساز یونیتی برای خروجی‌های روی محیط واقعی دقت درست‌تری دارد و این آزمایشات روی خروجی واقعی انجام شده است.

جدول ۱ نام و تعداد رئوس اشیاء را با دقت ۱۰۰ راس نشان می‌دهد. این اعداد با استفاده از ابزار آمار یونیتی استخراج شده‌اند.

آزمایشات زیر روی دستگاهی با مقدار رم ۱۶ گیگابایت و واحد پردازنده مرکزی Intel Core i7 7700HQ انجام شده است.

[1] Profiler

جدول ۷–۱ تعداد رئوس اشکال مورد آزمایش

نام	تعداد رئوس
صفحه	1.2k
مکعب	800
کره	2.3k
استوانه	1.2k
کره پیچیده	25k

در ادامه آزمایش خود را برای دو روش کلیک و صفحه بُرنده روی اشیای ذکر شده به همراه توپر سازی، انجام می‌دهیم. نتایج این آزمایش در جدول ۲ قابل رویت است.

جدول ۷–۲ نتایج آزمایش با دو روش کلیک و صفحه به همراه توپر سازی جسم مورد برش

	روش صفحه		روش کلیک	
نام	زمان (میلی‌ثانیه)	نرخ فریم	زمان (میلی‌ثانیه)	نرخ فریم
صفحه	1.48	+90	2.86	+90
مکعب	2.66	+90	2.41	+90
کره	3.53	+90	5.31	+90
استوانه	1.91	+90	3.12	+90
کره پیچیده	15.04	+60	16.84	≈60

همچنین برای برش اشیاء بدون توپر سازی، نتایج در جدول ۳ درج شده است.

جدول ۷–۳ نتایج آزمایش با دو روش کلیک و صفحه بدون توپر سازی جسم مورد برش

	روش صفحه		روش کلیک	
نام	زمان (میلی‌ثانیه)	نرخ فریم	زمان (میلی‌ثانیه)	نرخ فریم
صفحه	2.56	+90	3.17	+90
مکعب	0.29	+90	1.01	+90
کره	3.90	+90	3.96	+90
استوانه	1.12	+90	1.30	+90
کره پیچیده	15.31	+60	16.13	+60

دلیل اینکه صفحه در حالت توخالی زمان بیشتری می‌برد این است که در حالت توپر سازی چیزی پر نمی‌شود؛ زیرا صفحه بعد از برش فضای خالی برای پرسازی بوجود نمی‌آورد. این در حالی است که در حالت توخالی، پشت صفحه نیز ساخته می‌شود.

همانطور که مشاهده می‌شود حتی برای کره پیچیده که بیش از ۲۵ هزار راس دارد می‌توانیم به نرخ فریم قابل توجهی برسیم.

فصل ۸
جمع‌بندی و پیشنهادات بهبود

جمع بندی و پیشنهادات بهبود

در این فصل جمع‌بندی خلاصه‌ای از کارهای انجام شده خواهیم داشت و در ادامه پیشنهاداتی برای بهبود چارچوب ساخته شده ارائه می‌شود.

۸-۱-جمع بندی و نتیجه گیری

هدف از انجام این پروژه، ساخت یک چارچوب قابل استفاده مجدد برای توسعه‌دهندگان بازی بود که با آن بتوانند پایه‌ای کارامد از لحاظ کارایی و سودمندی برای پیاده‌سازی مکانیزم برش اشیاء داشته باشند. علاوه بر این نحوه‌ی نگارش کد آن را برای تغییر آماده کرده است تا توسعه‌دهندگان عملکرد-های شخصی‌سازی شده خود را به آن اضافه کنند.

توسعه‌دهندگان زیادی برای ساخت بازی‌های ویدئویی از موتور بازی‌سازی یونیتی استفاده می‌کنند؛ لذا تصمیم گرفته شد این چارچوب با استفاده از این موتور ساخته شود و بصورت متن باز در اختیار توسعه‌دهندگان قرار گیرد.

با توجه به همه‌منظوره بودن این ابزار، گسترش بیشتر، باعث می‌شود که این چارچوب با خاص شدن، برای همه‌ی توسعه دهندگان قابل استفاده نباشد، گرچه الگوریتم‌ها و روش‌های جایگزین معرفی و بررسی شدند و روش جدیدی برای چالش‌های این مسئله پیشنهاد شد.

۸-۲-پیشنهادات

در ادامه پیشنهاداتی راجع به این پروژه مطرح می‌شود که باعث بهبود آن در زمینه‌های مختلف می-شود. این پیشنهادات می‌توانند در صورت کلی بودن به خود چارچوب اضافه شوند، یا بصورت افزونه اختیاری برای کاربردهای خاص در کنار آن قرار گیرند.

۸-۲-۱-پشتیبانی از توری های پیچیده تر

پیاده‌سازی برش توری‌های مقعر در صورتی که نیاز به برش اشیای پیچیده‌تر داشته باشیم اولین نیاز این چارچوب است. بسیاری از بازی‌ها فقط از برش اشیای محدب استفاده می‌کنند ولی اینکه طراحان بازی محدود به ابزار مورد استفاده خود باشند، خلاقیت بازی‌سازها را کاهش می‌دهد و پذیرفته نیست. به علاوه ابزار تولید شده فقط زیرتوری اصلی توری‌ها را قطع می‌کند. البته در اکثر بازی‌ها توری‌ها فقط یک زیرتوری دارند، منتها با این کار مطمئن می‌شویم قادر به برش تمام توری-های ساده هستیم.

۸-۲-۲-استفاده از جنس جدید در محل برش

اشیای برش خورده می‌توانند در محل برش جنس جدیدی داشته باشند. برای مثال یک پرتقال می‌تواند بعد از برش خوردن درون متفاوتی از هندسه بیرونی خود داشته باشد. باید توجه داشت با این کار زیرتوری جدید معرفی می‌شود و برای اینکه برش‌های بوجود آمده نیز قابلیت برش خوردن را داشته باشند، باید ابزار از برش زیرتوری‌های بیشتر پشتیبانی کند.

۸-۲-۳-پشتیبانی از اشیای استخوان بندی شده

اجسام استخوان بندی شده در بازی‌های ویدئویی استفاده زیادی دارند. برای مثال انسان‌ها و حیوانات که دارای انیمیشن هستند استخوان‌بندی شده اند. برای برش این اشیاء علاوه بر برش توری نیاز به برش استخوان‌بندی هم داریم. ابزار توسعه داده می‌توان با پختن توری استخوان‌بندی شده، آن را در لحظه برش تبدیل به توری عادی کند که قابل برش دادن باشد، در این صورت بعد از برش انیمیشن‌های طراحی شده اجرا نمی‌شوند. در صورتی که پس از برش بخواهیم دو تکه برش خورده بصورت پویا (انیمیشن یا عروسک پارچه‌ای) در محیط حرکت کنند، می‌توانیم با نگه‌داشتن استخوان برای دو تکه، فقط مثلث‌های نامربوط را از دو تکه برش حذف و مثلث‌های جدید را اضافه کنیم، در این صورت می‌توان به چنین ویژگی‌ای دست یافت.

۸-۲-۴-انجام پاکسازی توری

پاکسازی توری یک مسئله پیچیده است ولی با استفاده از یک روش پاکسازی به عنوان پیش‌پردازش می‌توان با کاهش تعداد مثلث‌های پردازش شده، به کارایی بهتری رسید. باید توجه داشت که پیچیدگی این روش پاکسازی نباید بیشتر از پیچیدگی الگوریتم برش باشد تا این پاکسازی ارزش پیاده‌سازی داشته باشد.

۸-۲-۵-برش های پیچیده

استفاده از صفحات پیچیده‌تر یا حتی اشیای سه‌بعدی برای برش گرچه در بازی‌های ویدئویی کاربرد چندانی ندارد، ولی به قطع مسئله‌ی چالش برانگیزی است. برای مثال اگر بخواهیم یک جسم را بصورت قوس دار برش بزنیم، می‌توان قوس را به چند صفحه‌ی محدود تقسیم کرد و عملیات برش را انجام داد.

۸-۲-۶-برش همگام

با استفاده از برش همگام می‌توانیم مطمئن شویم که اگر عملیات برش به طول بیانجامد، بازی متوقف نمی‌شود و فقط نتیجه‌ی برش با تاخیر دیده می‌شود. این تاخیر را به سادگی می‌توان با جلوه‌های بصری یا طراحی بازی پوشاند. برای مثال با آهسته کردن بازی می‌توان زمان بیشتری برای برش خرید، بدون اینکه توقف یا افت فریم در نتیجه‌ی نهایی دیده شود.

۸-۲-۷-موازی سازی

برای کاربردهای در مقیاس خیلی بزرگ، مثل بازی‌های با بودجه بالا، می‌توان از موازی‌سازی بهره برد. این‌کار می‌تواند استفاده از ریسمان‌های واحد پردازنده مرکزی و یا هسته‌های واحد پردازنده گرافیکی باشد. البته باید توجه داشت که سربار گذر داده‌های از یک سخت‌افزار به سخت‌افزار دیگر نسبت به منفعت بدست آمده کمتر باشد.

منابع و مراجع

[1] J. H. a. S. Nichols, "Measuring Stress Causes, Experiences and Outcomes Worldwide," 15 April 2021. [Online]. Available: https://news.gallup.com/opinion/gallup/347309/measuring-stress-causes-experiences-outcomes-worldwide.aspx.

[2] K. Kaye, "Impact of COVID-19 on consumer spending: Booze buying and video games are up; travel, lodging sink," 26 March 2020. [Online]. Available: https://www.geekwire.com/2020/impact-covid-19-consumer-spending-booze-buying-video-games-travel-lodging-sink/.

[3] L. Reinecke, "Games and Recovery: the Use of Video and Computer Games to Recuperate From Stress and Strain," *Journal of Media Psychology: Theories, Methods, and Applications,* vol. 21, pp. 126-142, 2009.

[4] New Zoo, "Key Numbers," 2021. [Online]. Available: https://newzoo.com/key-numbers/. [Accessed October 2021].

[5] E. S. a. A. M. García, "Intercultural Perspective on Impact of Video Games on Players: Insights from a Systematic Review of Recent Literature," *Educational Sciences: Theory & Practice,* 2020.

[6] P. Palandrani, "Video Games & Esports: Building on 2020’s Rapid Growth," Global Xetfs, 9 March 2021. [Online]. Available: https://www.globalxetfs.com/video-games-esports-building-on-2020s-rapid-growth/. [Accessed October 2021].

[7] "Unity Asset Store," Unity Games, [Online]. Available: https://assetstore.unity.com/.

[8] Halfbrick Studios, "Fruit Ninja®," Halfbrick Studios.

[9] "Sword Play! Ninja Slice Runner," CASUAL AZUR GAMES.

[10] "Fruit Ninja® Google Play," [Online]. Available: https://play.google.com/store/apps/details?hl=en-ca&id=com.halfbrick.fruitninjafree.

[11] Steam Spy, "METAL GEAR RISING: REVENGEANCE," [Online]. Available: https://web.archive.org/web/20180411214550/steamspy.com/app/235460.

[12] J. Dunning, "PlayStation Awards See Grand Theft Auto V Take Home the Platinum Prize," Playstation Lifestyle, 3 December 2013. [Online]. Available: https://www.playstationlifestyle.net/2013/12/03/playstation-awards-see-grand-theft-auto-v-take-home-the-platinum-prize/. [Accessed October 2021].

[13] A. Garst, "Video game development in Iran: Limited tools, front companies and a specter of war," The Washington Post, 5 Febraury 2020. [Online]. Available: https://www.washingtonpost.com/video-games/2020/02/05/video-game-development-iran-limited-tools-front-companies-specter-war/. [Accessed October 2021].

[14] S. Stewart, "What Is The Best FPS For Gaming?," Gaming Scan, 5 Febraury 2021. [Online]. Available: https://www.gamingscan.com/best-fps-gaming/. [Accessed October 2021].

[15] "Unity (game engine)," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine).

[16] D. Gajsek, "Unity vs Unreal Engine for XR Development: Which One Is Better?," Circuit Stream, 16 March 2021. [Online]. Available: https://circuitstream.com/blog/unity-vs-unreal/. [Accessed October 2021].

[17] Unity Games, "Unity - Our Company," Unity Games, 2020. [Online]. Available: https://unity.com/our-company. [Accessed October 2021].

[18] TNW Deals, "This engine is dominating the gaming industry right now," TNW News, 24 March 2016. [Online]. Available:

https://thenextweb.com/news/engine-dominating-gaming-industry-right-now. [Accessed October 2021].

[19] Unity Games, "Unity Analytics," Unity Games, [Online]. Available: https://unity.com/features/analytics.

[20] J. Miller, "Using .NET 4.x in Unity," Microsoft, 29 August 2018. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/gamedev/unity/unity-scripting-upgrade. [Accessed October 2021].

[21] J. Dunstan, "Basic LINQ Performance," 16 March 2015. [Online]. Available: https://www.jacksondunstan.com/articles/2994. [Accessed October 2021].

[22] "High-Level Shading Language," Wiki[edia, [Online]. Available: https://en.wikipedia.org/wiki/High-Level_Shading_Language.

[23] D. Arayan, "Ezy Slice," Github, [Online]. Available: https://github.com/DavidArayan/ezy-slice. [Accessed October 2021].

[24] Epic Games, "Slice Procedural Mesh," Epic Games, [Online]. Available: https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Components/ProceduralMesh/SliceProceduralMesh/.

[25] Stas Bz, "Mesh Slicer," [Online]. Available: https://assetstore.unity.com/packages/tools/modeling/mesh-slicer-59618.

[26] O. S. Cajaraville, "Four Ways to Create a Mesh for a Sphere," 7 December 2015. [Online]. Available: https://medium.com/@oscarsc/four-ways-to-create-a-mesh-for-a-sphere-d7956b825db4. [Accessed October 2021].

[27] J. d. Vries, "Normal Mapping," Learn OpenGL, [Online]. Available: https://learnopengl.com/Advanced-Lighting/Normal-Mapping. [Accessed October 2021].

[28] LMHPOLY, "HOW TO CHANGE OBJECT SHADING IN UNITY," LMHPOLY, 24 October 2018. [Online]. Available: https://www.lmhpoly.com/tutorials/how-to-change-object-shading-in-unity. [Accessed October 2021].

[29] M. S. a. K. Akeley, "The OpenGL Graphics System: A Specification (Version 4.0 - Core Profile)," The Khronos Group Inc, 2010.

[30] Unity Games, "MeshTopology," Unity Games, 2020. [Online]. Available: https://docs.unity3d.com/ScriptReference/MeshTopology.html.

[31] Unity Games, "Occlusion culling," Unity Games, 2020. [Online]. Available: https://docs.unity3d.com/Manual/OcclusionCulling.html.

[32] M. M., "Types of Polygons," Tutors.com, [Online]. Available: https://tutors.com/math-tutors/geometry-help/types-of-polygons.

[33] CalcWorkshop, "Classifying Polygons," CalcWorkshop, 21 January 2020. [Online]. Available: https://calcworkshop.com/quadrilaterals/classifying-polygons/. [Accessed October 2021].

[34] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Processing Letters,* vol. 2, pp. 18-21, 1973.

[35] A. M. Andrew, "Another Efficient Algorithm for Convex Hulls in Two Dimensions," *Information Processing Letters,* vol. 9, pp. 216-219, 1979.

[36] R. Graham, "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set," *Information Processing Letters,* vol. 1, no. 4, pp. 132-133, 1972.

[37] A. C.-C. Yao, "A Lower Bound to Finding Convex Hulls," *Journal of the ACM,* vol. 28, no. 4, p. 780–787, 1981.

[38] M. d. Berg, M. v. Kreveld, M. Overmars and O. Schwarzkopf, "3: Polygon Triangulation," in *Computational Geometry*, Springer-Verlag, 2000, pp. 45-61.

[39] H. ElGindy, H. Everett and G. T. Toussaint, "Slicing an ear using prune-and-search," *Pattern Recognition Letters,* vol. 14, no. 9, pp. 719-722, 1993.

[40] G. H. Meisters, "Polygons have ears," *American Mathematical Monthly,* vol. 82, no. 6, pp. 648-651, 1975.

[41] R. E. Tarjan and C. J. Van Wyk, "An O(n log log n)-time algorithm for triangulating a simple polygon," *SIAM Journal on Computing,* vol. 17, no. 1, p. 143–178, 1988.

[42] K. L. Clarkson, R. Tarjan and C. J. van Wyk, "A fast Las Vegas algorithm for triangulating a simple polygon," *Discrete & Computational Geometry,* vol. 4, no. 5, p. 423–432, 1989.

[43] R. Seidel, "A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons," *Computational Geometry,* vol. 1, p. 51–64, 1991.

[44] K. L. Clarkson, R. Cole and R. E. Tarjan, "Randomized parallel algorithms for trapezoidal diagrams," *International Journal of Computational Geometry & Applications,* vol. 2, no. 2, p. 117–133, 1992.

[45] B. Chazelle, "Triangulating a Simple Polygon in Linear Time," *Discrete & Computational Geometry,* vol. 6, no. 3, p. 485–524, 1991.

[46] J. d. Vries, "Transformations," Learn OpenGL, [Online]. Available: https://learnopengl.com/Getting-started/Transformations. [Accessed October 2021].

[47] Unity Games, "Compute shaders," Unity Games, 2020. [Online]. Available: https://docs.unity3d.com/Manual/class-ComputeShader.html. [Accessed October 2021].

[48] Unity Games, "OverlapBox," Unity Games, 2020. [Online]. Available: https://docs.unity3d.com/ScriptReference/Physics.OverlapBox.html. [Accessed October 2021].

[49] Lexdev, "Overwatch Shield," 2019. [Online]. Available: .net/tutorials/case_studies/overwatch_shield.html. [Accessed October 2021].

پیوست آ

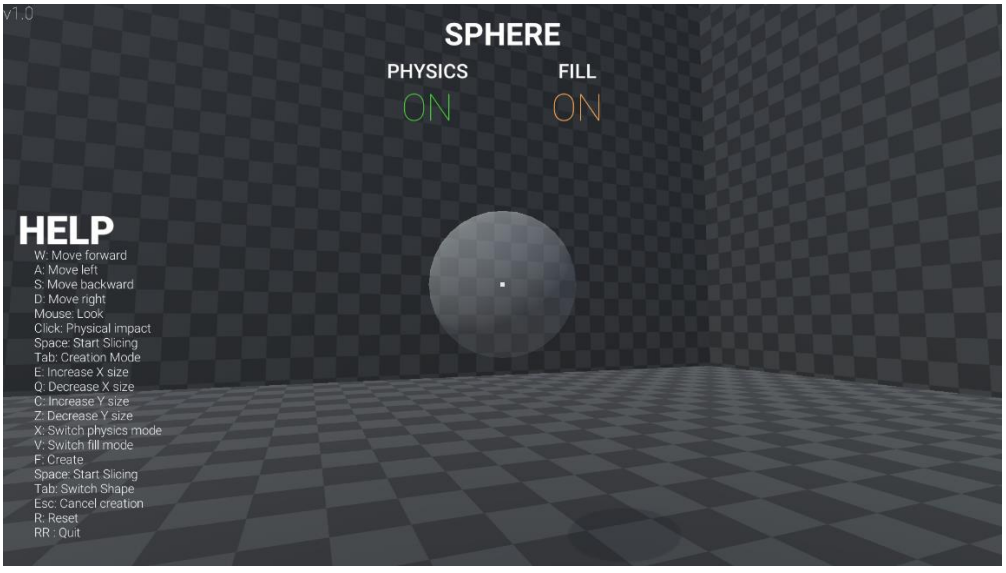
راهنمای کار با دمو

راهنمای کار با دمو

در این قسمت، راهنمای جامعی درباره نحوه کار با خروجی که در اختیارتان قرار گرفته است ارائه می‌شود. خروجی به حجم ۶۳ مگابایت حاوی یک فایل قابل اجرا exe. در ویندوز با نام Wakizashi.exe است.

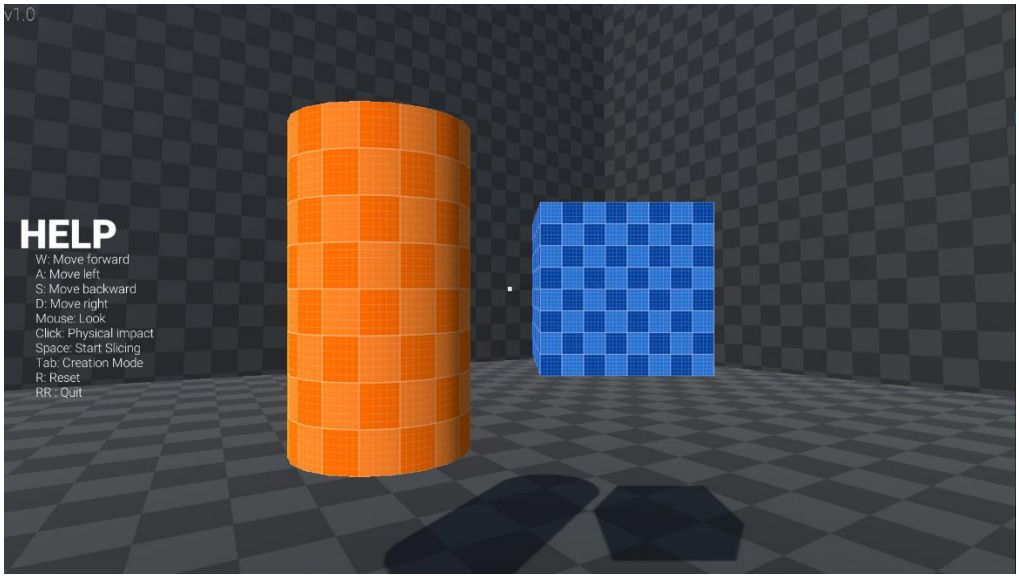
در ابتدا این فایل را باز کرده و منتظر بمانید تا صحنه‌ی بازی بارگیری شود. در سمت چپ صفحه راهنمای کلیدها دیده می‌شود که می‌توانید از آن کمک بگیرید. برای خاموش و روشن کردن راهنما می‌توانید از دکمه F1 استفاده کنید. این راهنما بصورت حساس به زمینه کار می‌کند و دکمه‌های پیشنهاد شده وابسته به حالت ویرایش دمو است.

در حالت پیشفرض می‌توانید در محیط قدم بزنید که این کار با کمک دکمه‌های W,A,S,D انجام می‌گیرد. برای ورود به حالت اضافه کردن اجسام، کلید Tab را فشار دهید. با فشردن دوباره این دکمه، جسم ساخته شده تغییر می‌کند که پیش‌نمایش آن بصورت کم‌رنگ در صفحه دیده می‌شود. می‌توانید با دکمه‌های E,Q,C,Z اندازه جسمی که قصد ساخت آن را دارید عوض کنید و با حرکت و نگاه کردن به اطراف، که با موس انجام می‌شود، موقعیت آن را تنظیم کنید. همچنین برای تنظیم اینکه جسم توپر (نارنجی) یا توخالی (آبی) باشد کلید V و برای فعال‌سازی فیزیک پس از برش یا خاموش کردن آن کلید X را فشار دهید.



شکل آ-۱ حالت ساخت مدل

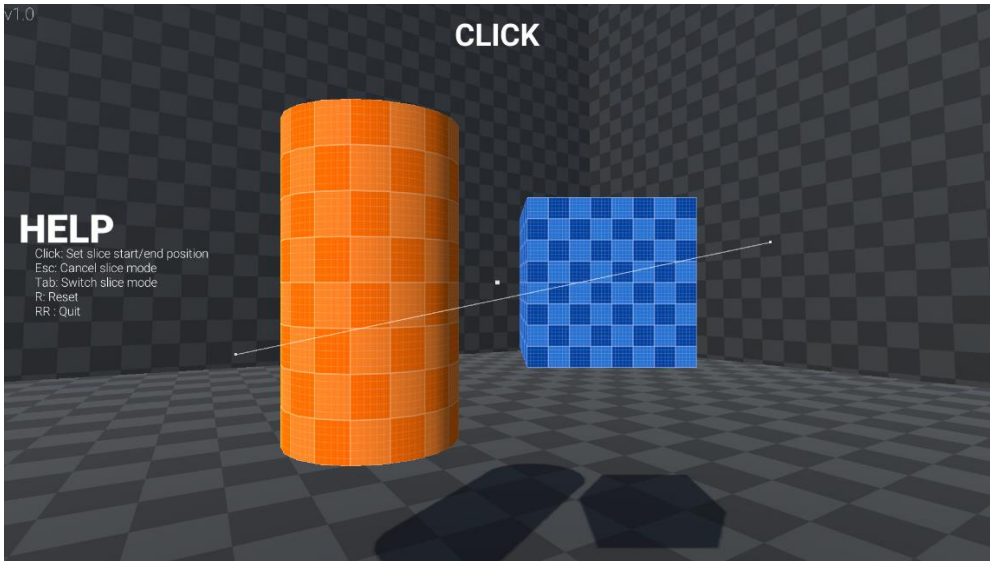
پس از تنظیم مکان و اندازه جسم مورد نظر کلید F را فشار دهید تا آن جسم به محیط اضافه شود. برای خروج از حالت ساخت اشیاء، کلید Esc را فشار دهید. برای حذف تمامی اشیای اضافه شده کلید R را فشار دهید.



شکل آ- ۲ حالت حرکت آزاد

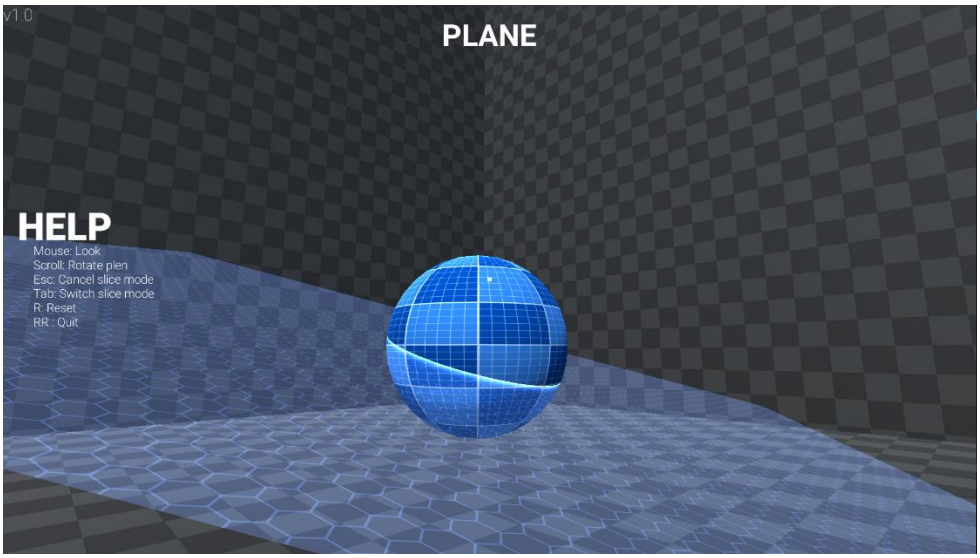
برای اینکه با حالت برش وارد شوید ابتدا مکان و زاویه دید خود را تنظیم کنید و سپس دکمه Space را بزنید. در این حالت صفحه کیبورد قفل می‌شود که در گوشه‌ی پایین سمت راست صفحه قابل دیدن است.

در حالت Click موس شما باید روی صفحه ظاهر شود و با کلیک نقطه شروع و پایان برش را انتخاب کنید.



شکل آ-۳ برش با کلیک

با زدن دکمه Tab به حالت Plane وارد می‌شود و صفحه برنده در صفحه نمایش شما نشان داده می‌شود. با کلیک موس می‌توانید برش را اعمال کنید. همینطور برای چرخاندن صفحه می‌توانید از چرخ موس استفاده کنید. برای خروج از حالت برش کلید Esc را فشار دهید.



شکل آ-۴ برش با صفحه

برای اینکه از برنامه خارج شوید دوبار پشت سر هم کلید R را فشار دهید. برای اینکه بتوانید بهتر اشیای برش خورده را مورد بررسی قرار دهید با کلیک روی مرکز صفحه می‌توانید به آن‌ها نیروی فیزیکی وارد کنید.