

دانشکده مهندسی کامپیوتر

**پیاده​سازی یک موتور فیزیک با زبان جاوا اسکریپت جهت توسعه بازی​های تحت وب در قالب HTML5**

پایان‌نامه برای دریافت درجه کارشناسی

**کامران نوبهار**

86522095

استاد راهنما:

دکتر بهروز مینایی

دی ماه 1391

به نام خدا

**چکیده**

در این گزارش به بررسی جدیدترین امکانات معرفی شده در HTML5 و همچنین نحوه پیاده​سازی یک موتور فیزیک جهت استفاده از یکی از مهم​ترین عناصر جدید HTML یعنی canvas می​پردازیم.

در ابتدا با معرفی مهم​ترین تکنولوژی​های وب، تاریخچه پیدایش، رشد و مقبولیت آن​ها به بررسی بسترهای ظهور عناصر جدید HTML5 پرداخته و جایگاه حال و آینده این عناصر و همچنین گزینه​های پیشینشان را مطالعه​​ای کلی می​کنیم. همچنین در مقدمه این مسئله مطرح می​شود که چرا canvas به عنوان یکی از عناصر تازه در HTML به عنوان رکن اصلی این پروژه برای هدف قرار گرفتن کل پیاده​سازی انتخاب شده است. به طور کلی می​توان گفت که canvas پس از ظهور و حکم فرمایی تعداد متنوعی از ساختارهای نرم​افزاری برای ارائه محتوای چندرسانه​​ای می​تواند به عنوان جایگزینی جامع و استاندارد استفاده شود و در این پروژه نشان داده می​شود که چطور یک موتور فیزیک برای ایجاد بازی​هایی در این ساختار می​تواند ساخته شود و کار کند.

Canvas اما یک محیط گرافیکی برداری است که در درون یک سند HTML5 جای می​گیرد و برای دسترسی به و دستکاری در آن می​بایست از درخت DOM سند استفاده کرده و با JavaScript با کار با آن بپردازیم. رابط کاربری معرفی شده با canvas یک context دو بعدی در اختیار می​گذارد که با رابط کاربری آن می​توانیم انواع اشکال برداری را در Canvas رسم نماییم. این رابط کاربری شامل انواع توابع مختلف و متنوع برای ترسیم هر شکل، تصویر و یا حتی دستکاری​های پیکسلی می​باشد. در فصل یک به معرفی این توابع و قابلیت​ها و همچنین دیگر مطالب فنی مرتبط می​پردازیم.

Canvas از آن جایی که عنصر بسیار جدیدی در وب است، ابزارها و کتابخانه​های کافی برای کار و استفاده با آن هنوز وجود ندارد، یکی از این کمبودها یک موتور فیزیک جامع برای ساخت و ایجاد بازی​های دوبعدی است. موتور فیزیک در واقع یک قطعه کد است که وظیفه محاسبات مربوط به شبیه‌سازی فیزیکی اشیا درون یک بازی با اشیا ما به ازا آن​ها در دنیای واقعی را دارد. ضرورت ایجاد این موتور فیزیک به طور جدا از ضرورت کلی وجود آن نیست و چنان که بعدا در فصل دو بیشتر توضیح خواهیم داد صرفه جویی در کد و بالا بردن کیفیت بازی​های کامپیوتر دو دلیل عمده ضرورت ایجاد این موتور است.

در این پروژه ابتدا یک موتور فیزیک ذره​​ای و سپس با گسترش آن یک موتور فیزیک تجمیع ذرات را پیاده​سازی می​کنیم که در فصل سه و چهار به معرفی ویژگی​های آن و همچنین مبانی اولیه ریاضی و فیزیک مربوط به آن می​پردازیم و آن چه را که در موتور قصد شبیه‌سازی داریم مطالعه خواهیم کرد.

برای گسترش موتور فیزیک سپس بحث نیروها و مولدهای نیرو پیش کشیده می​شود و نیروهای مختلف نیز باید در موتور گنجانده شود که در فصل بعد به آن پرداخته می​شود. همچنین مهمترین موجودیت موتور تجمیع ذرات که محدودیت​ها هستند نیز در موتور ضروری می​باشند. این موجودیت​ها نیز در فصل هفت و هشت بررسی و نحوه پیاده​سازی آن​ها بیان می​شود.

در انتهای این گزارش خواهیم دید که چطور در این پروژه یک موتور فیزیک تجمیع ذرات با زبان جاوا اسکریپت برای استفاده در محیط canvas پیاده​سازی شده است.

**فهرست مطالب**

[فصل اول: تکنولوژی​های بستر 9](#_Toc345323366)

[1.1 مقدمه 10](#_Toc345323367)

[2.1 تکنولوژی​ها 10](#_Toc345323368)

[3.1 جمع​بندی و نتیجه​گیری 20](#_Toc345323369)

[فصل دوم: بررسی فیزیک بازی 22](#_Toc345323370)

[1.2 مقدمه 23](#_Toc345323371)

[2.2 موتور فیزیک 24](#_Toc345323372)

[3.2 نوع و روش پیاده​سازی موتور فیزیک 26](#_Toc345323373)

[4.2 جمع​بندی و نتیجه​گیری 27](#_Toc345323374)

[فصل سوم: فیزیک ذرات 29](#_Toc345323375)

[1.3 مقدمه 30](#_Toc345323376)

[2.3 ریاضیات ذرات 30](#_Toc345323377)

[3.3 جمع​بندی و نتیجه​گیری 33](#_Toc345323378)

[فصل چهارم: قوانین حرکت 34](#_Toc345323379)

[1.4 مقدمه 35](#_Toc345323380)

[2.4 ذره 35](#_Toc345323381)

[3.4 دو قانون اول 35](#_Toc345323382)

[4.4 اعمال معادلات 38](#_Toc345323383)

[5.4 جمع​بندی و نتیجه​گیری 40](#_Toc345323384)

[فصل پنجم: استفاده از فیزیک ذره​​ای در موتور 41](#_Toc345323385)

[1.5 مقدمه 42](#_Toc345323386)

[2.5 پرتابه​ها 42](#_Toc345323387)

[3.5 آتش‌بازی 45](#_Toc345323388)

[4.5 جمع​بندی و نتیجه​گیری 49](#_Toc345323389)

[فصل ششم: فیزیک تجمیع اجرام، نیروهای عمومی 51](#_Toc345323390)

[1.6 مقدمه 52](#_Toc345323391)

[2.6 اصل دالامبر 52](#_Toc345323392)

[3.6 تامین‌کنندگان نیرو 53](#_Toc345323393)

[4.6 جمع​بندی و نتیجه​گیری 59](#_Toc345323394)

[فصل هفتم: فنرها و خواص فنری 61](#_Toc345323395)

[1.7 مقدمه 62](#_Toc345323396)

[2.7 قانون هوک 62](#_Toc345323397)

[3.7 مولد​های نیروی فنرگونه 63](#_Toc345323398)

[4.7 جمع​بندی و نتیجه​گیری 76](#_Toc345323399)

[فصل هشتم: محدودیت​های قوی 77](#_Toc345323400)

[1.8 مقدمه 78](#_Toc345323401)

[2.8 تحلیل ساده برخورد 78](#_Toc345323402)

[3.8 پدیده​های برخوردگونه 107](#_Toc345323403)

[4.8 جمع​بندی و نتیجه​گیری 114](#_Toc345323404)

[فصل نهم: استفاده از موتور فیزیک تجمیع جرم 115](#_Toc345323405)

[1.9 مقدمه 116](#_Toc345323406)

[2.9 نمای کلی موتور 116](#_Toc345323407)

[3.9 جمع​بندی و نتیجه​گیری 118](#_Toc345323408)

[نتیجه‌گیری و پیشنهادها 122](#_Toc345323409)

[منابع و مآخذ 126](#_Toc345323410)

**فهرست شکل​ها**

[شکل 1 تنظیمات نوشته در canvas 14](#_Toc345338511)

[شکل 2 کشیدن خط در canvas 16](#_Toc345338512)

[شکل 3 تصویر در canvas 18](#_Toc345338513)

[شکل 4 مثال آتش بازی 49](#_Toc345338514)

[شکل 5 مثال مولدهای نیرو 60](#_Toc345338515)

[شکل 6 فنرها 66](#_Toc345338516)

[شکل 7 مثال دوم فنرها 67](#_Toc345338517)

[شکل 8 نیروهای شناوری 69](#_Toc345338518)

[شکل 9 مثال اول نیروهای شناوری 73](#_Toc345338519)

[شکل 10 مثال سوم نیروهای شناوری 74](#_Toc345338520)

[شکل 11مثال دوم نیروهای شناوری 75](#_Toc345338521)

[شکل 12مثال چهارم نیروهای شناوری 76](#_Toc345338522)

[شکل 13 خطای نرمال 84](#_Toc345338523)

[شکل 14 تداخل 88](#_Toc345338524)

[شکل 15 مشکل تداخل در شکل​های بزرگ 90](#_Toc345338525)

[شکل 16 مشکل برخوردهای ایستا 93](#_Toc345338526)

[شکل 17مشکل اولویت تداخل​ها 98](#_Toc345338527)

[شکل 18 تحلیل برخورد با ثابت متفاوت 105](#_Toc345338528)

[شکل 19برخورد با جرم​های متفاوت 106](#_Toc345338529)

[شکل 20 بعد از برخورد 107](#_Toc345338530)

[شکل 21مثال اول طناب‌ها 109](#_Toc345338531)

[شکل 22 مثال دوم طناب‌ها 110](#_Toc345338532)

[شکل 23 مثال میله​ها 112](#_Toc345338533)

[شکل 24 مثال آونگ 114](#_Toc345338534)

[شکل 25 مثال ترکیبی 114](#_Toc345338535)

[شکل 26 مثال تجمیع اجرام 119](#_Toc345338536)

[شکل 27 انواع اجرام تجمیعی 120](#_Toc345338537)

[شکل 28 یک بازی با موتور 121](#_Toc345338538)

**مقدمه**

گستره جهانی وب به عنوان مهم​ترین بستر انتقال اطلاعات و داده و همچنین ارتباطات دیگر در شبکه اینترنت شناخته می​شود، آن چنان که انواع ارتباطات دیجیتال چه از نوع انتقال صرف اطلاعات و یا ارتباطات تعاملی را پشتیبانی می​کند. مرورگران وب مهمترین و در واقع رکن اصلی در این ارتباط می​باشند و بستر نمایش و تعامل با انواع داده​های وب در اینترنت را فراهم می​آورند. در سال 1980 استانداردی برای انتقال اسناد در شبکه​های کامپیوتری مرکز CERN تدوین شد تا انتقال اطلاعات بتواند در یک قالب مشخص و کارآمد انجام بگیرد. این استاندارد که HTML نام گرفت بعدا به مهمترین زبان تدوین اطلاعات در وب بدل شد، و می​توان گفت که هم اکنون اکثر داده​ها در وب در شبکه اینترنت با این زبان انتقال می​یابند.

HTML که در واقع زبان نشانه‌گذاری ابرمتنی است، در ابتدا مهمترین هدفش تدوین ساختاری بود که کاربر بتواند به راحتی در میان اسناد وب گردش کند و از یک به متن به متن دیگری رجوع کند، اما کم کم با پیشرفت تکنولوژی​های انتقال اطلاعات و به تبع آن بالارفتن هر چه بیشتر پهنای باند اینترنت امکان این به وجود آمد تا داده​های دیگری را نیز بتوان از طریق وب جابجا کرد، که از آن جمله می​توان به عکس​ها، فیلم​ها، فایل​های صوتی و غیره اشاره کرد. در نتیجه لازم بود تا زبان HTML بتواند این نوع داده​ها را نیز در خود جای دهد، تا بتوان آن​ها در کامپیوترهای کاربری دریافت و مشاهده کرد. نوع و شکل داده​های چند رسانه​​ای این گونه کم کم زیاد شده و با حساب انواع فرمت​های هر رسانه اعم از فیلم، موسیقی و عکس تلاش​های زیادی لازم بود تا با استاندارد کردن آن​ها در قالب HTML و احیانا خارج از آن در بستر وب امکان دریافت و ارسالشان به طور گسترده به وجود آید. در این میان مرورگران وب نقشی حیاتی داشتند چرا که می​توان گفت تنها ابزار دریافت، نمایش و احیانا تعامل با وب در محیط کاربر بوده اند، و وظیفه پشتیبانی از انواع استانداردها و فرمت​های به وجود آمده و در حال به وجود آمدن برای عرضه به کاربر را بر عهده داشتند.

در میان انبوه شکل​های مختلف داده​های چند رسانه​​ای بعضی بیشتر مورد استقبال قرار می​گرفتند و بعضی کمتر همچنین کنسرسیوم گستره جهانی وب که مسئولیت استانداردسازی و پشتیبانی از استاندارهای وب را دارد، به بعضی انواع داده​ها بیشتر توجه کرده و به بعضی کمتر. برای مثال فرمت​های ویدئویی و صوتی در ابتدای عرضه شان در وب و تا چند سال پیش تنها از طریق برنامه​های جانبی دیگری قابل پخش بودند که باید در محیط کاربر نصب می​شدند. روش انتقال این داده​ها در نتیجه به این شکل می​بود که در صفحات فرستاده شده از طریق سرور وب به کاربر که با HTML استاندارد شده است، در درون سند، آدرس اینترنتی فایل صوتی یا تصویری داده می​شد تا از این طریق مرورگر بتواند با بارگذاری آن از آن آدرس فایل را دریافت کند، پس از دریافت فایل مرورگر با توجه به فرمت فایل افزونه مناسب را که قابلیت پخش و ارائه فایل را داشته و قبلا می​بایست نصب شده می​بود و به مرورگر الصاق می​شد فراخوانی کند، تا افزونه که معمولا با یک برنامه اصلی و جدا از مرورگر و نصب شده در محیط کاریر در ارتباط است فایل را در خود محیط مرورگر و در محلی از سند که توسط زبان HTML مشخص شده است پخش کند و نمایش دهد. برای مثال اگر فرمت ویدئویی و صوتی WMA را در نظر بگیریم، وقتی کاربری به صفحه​​ای از وب رجوع می​کند که حاوی فایلی از این فرمت است، مرورگر کاربر ابتدا چک می​کند که آیا افزونه​​ای را در اختیار دارد که بتواند این فایل را با کمک آن نشان دهد، و اگر نه پیغام خطایی را به او نشان می​دهد که افزونه مناسب فایل WMA باید نصب شود. فایل WMA فرمت اختصاصی برنامه Windows Media Player می​باشد و در محیط ویندوز تنها این برنامه قادر به پخش آن می​باشد، در نتیجه لازم است که اولا خود برنامه در محیط نصب شده و دوما افزونه Windows Media Player که برای مرورگر خاص مثلا Internet Explorer ساخته شده نیز نصب و به IE اضافه شود. به این صورت IE با فراخوانی آن افزونه و آن افزونه با فراخوان برنامه اصلی (WMP) فایل را در محل مناسب خود پخش می​کند.

در این میان دو تکنولوژی مهم دیگر در محیط وب رشد کردند. اول تکنولوژی اسکریپت نویسی در اسناد HTML بود که از این طریق امکان پویا کردن و تعاملی کردن این صفحات به وجود آمد. مهمترین زبان اسکریپت نویسی در این میان JavaScript بوده که امکانات بسیاری را در این زمینه به عرضه گذاشت. با قابلیت​های این زبان این امکان به وجود آمد که به صورتی برنامه ریزی شده بتوان به عناصر درونی اسناد HTML دسترسی داشت و آنها را تغییر داد، بدین صورت اولین امکان تعامل در طرف کاربر با سند وب به وجود آمده و سند وب می​توانست به عنوان یک برنامه کامپوتری ساده عمل کند و برای مثال از کاربر ورودی گرفته و پیغام مناسبی را چاپ کند. لازم به ذکر است که مانند تمام عناصر دیگر وب وظیفه اجرای اسکریپت​ها بر عهده مرورگر بوده اما بر خلاف عناصری چون Flash افزونه جداگانه​​ای برای اجرای آن لازم نمی​بود چرا که اسکریپت​ها زبانی​هایی هستند که خود مرورگر می​تواند تفسیر آنها را به عده بگیرد.

اما تکنولوژی مهم دوم که ابتدا اصلا در خارج از وب و اینترنت به وجود آمد، بستر Flash بود. Flash یک بستر و فرمت ارائه شده توسط شرکت Macromedia بود که برای ایجاد و نمایش انواع داده​های چند رسانه​​ای می​توانست به کار گرفته شود، تکنولوژی کشیدن اشکال بر پایه بردار​ها آن را تبدیل به قدرتمندترین فرمت شناخته شده در زمان خود برای ارائه داده چند رسانه​​ای کرد. در Flash همچنین بعدا امکان ایجاد انیمیشن​های برداری و کم کم با اضافه شدن قابلیت​های اسکریپت نویسی در درونش امکان ایجاد برنامه​های تعاملی از جمله بازی​های ویدئویی به وجود آمد. این امکانات که در ابتدا در محیط​های خارج از وب ارائه شده بودند در مقایسه با دیگر برنامه​هایی که امکان ایجاد بازی و انیمیشن را می​دادند برتری به نسبه​​ای به Flash دادند چنان که شرکت Macromedia تصمیم گرفت فایل​های با فرمت Flash را در اینترنت و در وب تیز ارائه کند. این ارائه منجر به یکی از بزرگترین تحولات در وب شد چنان که افزونه​​ای که برای نمایش Flash به کار می​رود تبدیل به مهمترین و با بیشترین تعداد استفاده از میان افزونه​های مرورگران وب شد. گستردگی Flash به گونه​​ای بود و می​توان گفت که هم اکنون نیز هست که بسیاری از وب سایت​های اینترنتی تمام محتوای خود را از طریق آن ارائه می​کنند و در اکثر وبسایت​های دیگر رد پای آن را می​توان یافت، از جمله استفاده گسترده در تبلیغات محیطی وبسایت​ها و یا ارائه داده​هایی که نیاز به انیمیشن دارند. راحتی استفاده و همچنین محیط زیبا و مورد پسند Flash و همچنین نبود رقیب جدی برای آن در محیط وب از جمله مهمترین عوامل موفقیت این تکنوژی در وب می​باشد. Flash حتی با پیشرفتی که خود داشته توانست نحوه ارائه فایل​های ویدئویی و صوتی را نیز متحول کند و پلیرهایی برای اجرای آن​ها ارائه دهد. یکی دیگر از کاربردهای وسیعی که Flash در محیط وب به وجود آورد امکان ایجاد بازی​های دو بعدی و حتی سه بعدی در این محیط بود، به طوری که حالا می​شد در یک سند HTML در محیط وب بتوان بازی کرد. هر چند این گونه بازی​ها در مقایسه با بازی​های ویدئویی که در محیط کامپیوتری و با کمک کارت​های گرافیک قدرتمند اجرا می​شوند ساده به نظر بیایند اما واقعیت این است که گستردگی و استفاده و محبوبیت بسیاری داشته‌اند. فراموش نباید کرد که Flash نیز مانند دیگر برنامه​های خارج از وب برای اجرا شدن در محیط مرورگر نیاز به یک افزونه برای مرورگر و همچنین یک برنامه اصلی در خارج از مرورگر دارد و به همان طریق پیشتر گفته شده اجرا می​شود.

نمایش فایل​های چند رسانه​​ای به نوع​هایی که توضیح داده شد در وب تا مدتی متدوال بود، اما کاربران اینترنتی کم کم متوجه شدند که بسیار روش ناکارآمدی است و مشکلات عدیده​​ای را به همراه دارد. از جمله این مشکلات می​توان به دشواری پشتیبانی محیط کاربری از تمام فرمت​های موجود با توجه به تنوع روز افزون انواع فایل​های چند رسانه​​ای اشاره کرد. به این صورت که کاربری که بخواهد انواع فایلها را در کامپیوتر خود از وب مشاهده کند می​بایست انواع افزونه​ها و برنامه​های مختلف را نصب کند و به محض برخورد با یک فرمت جدید مجبور خواهد شد تا یک افزونه جدید بارگذاری و نصب کند. مشکل دیگر خطرات عدیده امنیتی بود که از این طریق کاربران را تهدید می​کرد. افزونه​های ارائه شده برای مرورگرهای مختلف برای فرمت​های مختلف توسط شرکت​های گوناگونی ایجاد و عرضه می​شدند و با توجه به این که کنترلی بر روی محتوای فایل​های غیر از اسناد HTML نمی​تواند وجود داشته باشد، آسیب پذیری​های امنیتی زیادی می​تواند در افزونه​ها امنیت کاربران را به خطر اندازد. مرورگر هیچگونه کنترلی بر روی اجرای افزونه​ها نداشته و ارتباط با یک برنامه دیگر در محیط کاربری به شدت خطرناک می​باشد. فایل​های صوتی و تصویری می​توانند بالقوه فایل​های اجرای مخربی باشند که به راحتی از کنترل​های امنیتی مرورگر عبور کرده و از حفره​های امنیتی افزونه​ها استفاده کنند. بیشترین تعداد مشکلات امنیتی سال​های اخیر در محیط وب در رابطه با افزونه Flash بوده که هشدارهای جدی را در رابطه با این افزونه در پی داشته است. در واقع متن بسته بودن افزونه​ها و برنامه​هایی چون Flash همواره نگرانی کسانی که در پشتیبانی گستره جهانی وب نقش داشته اند را برانگیخته است. چنان که کنسریوم جهانی وب نیز همواره بر این نکته تاکید داشته است که روحیه جاری در محیط اینترنت بر متن باز بودن اطلاعات انتقال یافته و فایل​های اجرایی متکی است.

این موضوع شاید مهمترین دلیل تلاش​های بعدی این کنسرسیوم و افراد دیگری با همین دغدغه برای تدوین استانداردهای نو و یک پارچه برای نمایش و اجرای فایل​های چند رسانه​​ای در محیط وب بوده است. بدین صورت تلاش شد تا سایه افزونه​های گوناگون با مشکلات جدی از سر مرورگرها کم شود و خود آن​ها مسئول اجرا و نمایش فرمت​های گوناگون شوند. بدین صورت تمام تلاش​ها برای یک پارچه‌سازی و استانداردسازی این فرمت​ها در تدوین پنچمین نسخه HTML جمع شد و در آن به بار نشست. HTML5 که تغییرات بسیاری نسبت به نسخه پیشین خود کرد، بسیاری از مشکلات قبلی را حل نمود. اما تغییر مهمی​که در این جا مدنظر است اضافه شدن عناصر جدید audio، video و canvas می​باشد. عناصر audio و video همان طور که از نامشان معلوم است در جهت همان استانداردسازی فرمت​های صوتی و تصویری گفته شده ایجاد شده اند و فایل​های صوتی و تصویری را در خودشان حمل می​کنند به گونه​​ای که مرورگران وب بتوانند آن​هارا اجرا کنند، لازم به توضیح است که حالا در این استاندارد جدید وظیفه پشتیبانی از صوت و تصویر به عهده مرورگر قرار گرفته و نقش افزونه​ها حذف شده است.

اما عنصر اصلی مورد توجه ما Canvas می​باشد. عنصر canvas تکنولوژی بسیار جدیدی را به وب اضافه کرد که قابلیت ترسیم اشکال در یک سند وب را می​دهد. قابلیت​های این تکنولوژی را شاید بتوان با قابلیت​هایی مقایسه کرد که Flash در اختیار می​گذاشت. یعنی ترسیم اشکال، ایجاد انیمیشن و همچنین محیط​های تعاملی از جمله بازی​ها. Canvas در واقعی عنصری در یک سند HTML است که دارای مشخصاتی از جمله طول و عرض می​باشد. با این مشخصات یک محیط در اختیار قرار می​گیرد که می​توان با یک رابط کاربری از طریق JavaScript به آن دسترسی پیدا کرد و به ترسیم اشکال و ایجاد پویانمایی در ان پرداخت، همچنین به دلیل استفاده از script برای کار با آن امکان تعاملی کردن محیط نیز وجود داشته و به این صورت می​توان به طراحی بازی نیز در آن پرداخت. شاید بتوان گفت که یکی از انگیزه​های اساسی در ایجاد عنصر canvas به وجود آوردن جایگزینی برای Flash بوده تا با مقابله با مشکلاتی که افزونه​های Flash داشتند بتوان امکانات آن​ها را در محیط وب حفظ کرد به طوریکه بنای canvas از یکی از تکنولوژی​های ایجاد شده در شرکت Apple بوده که در واقع با حذف Flash از مرورگر Safari خود این عنصر را ابتدا معرفی کرده بود.

با توجه به این که عنصر canvas در محیط به نسبت بسیار جدید است و امکانات بسیار نویی را در اختیار می​گذارد بسیاری از کسانی که به کار پیاده​سازی در محیط وب مشغول بوده اند شروع به کار با این عنصر و تست کردن آن کردند، در این میان انیمیشن​ها و بازی​های متنوع و بسیاری ایجاد شد تا معلوم شود قدرت این ابزار تا چه حد می​باشد، از مهمترین پیاده​سازی​های موجود می​توان به پورت کردن بازی Quake توسط شرکت گوگل بر روی canvas و البته با استفاده از کتابخانه​های گرافیکی دیگر اشاره کرد. از آنجا که این بازی سه بعدی می​باشد مشخص شد که قدرت این ابزار تا چه حد بالاست.

در میان بازی​های ایجاد شده در canvas می​توان یک روند تکاملی را دید که در طول این مدت وجود داشته است، بازی​ها به تدریج غنی‌تر و دارای شکل​های پیچیده تری شده اند، و در این روند یکی از کمبودهایی که به نظر می​آید بسیار تعیین کننده می​باشد نبود یک موتور فیزیک خوب در این محیط است. بازی​های ویدئویی می​توانند بدون موتور فیزیک هم پیاده شوند اما بازی​هایی که نیاز داشته باشند تا پدیده​های فیزیکی و در طبیعت را در محیط کامپیوتر شبیه‌سازی کرده و تصور آن​هارا نشان دهند نیاز دارند که یا خود یک موتور فیزیک در درون بازی خود ایجاد کنند و یا از یک موتور فیزیک دیگر استفاده کنند. با توجه به ثابت بودن پدیده​های فیزیکی در دنیای واقعی و نیاز​های مشابهی که این گونه بازی​ها دارند می​توان نتیجه گرفت که طراحی و ایجاد یک موتور فیزیک جدا می​تواند به پیاده​سازی طیف وسیعی از بازی​های ویدئویی کمک کند، چنان که این بازی​های می​توانند بدون نیاز به نوشتن یک موتور فیزیک درونی برای خودشان جهت شبیه‌سازی پدیده​هایی که می​خواند در بازی خود به پدیده​های فیزیکی شبیه باشد از این موتور فیزیک استفاده کنند. پس روشن می​شود که چگونه در این محیط جدید ایجاد یکی موتور فیزیک که مختص این محیط است به کمک سیر پیشرفت پیاده​سازی​ها در این محیط می​انجامد.

در نتیجه در این پروژه قصد داریم پدیده​های فیزیکی دو بعدی قابل طرح در یک بازی را بررسی کرده و مدل​های ریاضی آن​ها را استخراج کنیم و با کمک آن​ها پیاده​سازی​های مناسبی برای هر پدیده فیزیکی انجام دهیم و درنتیجه یک موتور فیزیک دو بعدی ایجاد کنیم. مهمترین مسئله این موتور فیزیک زبان پیاده​سازی آن است که JavaScript می​باشد، زبانی اسکریپتی که قرار است در محیط مرورگر اجرا شود و در برنامه​هایی به کار آید که از canvas برای کشیدن شکل​های وابسته به پدیده​های فیزیکی استفاده می​کنند. در واقع چون برای کار با canvas لازم است که با JavaScript کار شود، پیاده​سازی موتور به این زبان ضروری می​نماید.

# فصل اول: تکنولوژی​های بستر

## مقدمه

برای پیاده​سازی موتور فیزیک نیاز داریم تا محیطی که قرار است بازی​های استفاده کننده در آن ایجاد شوند را به خوبی بشناسیم. در این فصل به مطالعه تکنولوژی​های بستر این پروژه در وب پرداخته و امکانات، قابلیت​ها و رابط​های کاربری مربوطه را بررسی می کنیم تا بتوانیم با توجه به آن​ها موتور فیزیک مناسب را بر پایه تکنولوژی​های هدف بنا کنیم.

## تکنولوژی​ها

HTML یک زبان نشانه‌گذاری است که اسناد را با توجه به موقعیت و نوع عناصرشان استانداردسازی می​کند، هر سندی که در محیط وب بخواهد قرار بگیرد و در مرورگر نشان داده شود لازم است که بازبان HTML نشانه‌گذاری شود. یک سند HTML نوعی به این صورت می​باشد:

<! DOCTYPE html>

<html>

<head>

<title>Sample page</title>

</head>

<body>

<h1>Sample page</h1>

<p>This is a <a href="demo.html">simple</a> sample.</p>

<!-- this is a comment -->

</body>

</html>

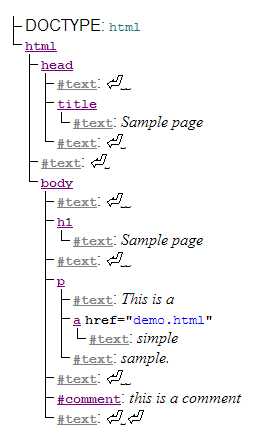
اسناد HTML شامل یک درخت از عناصر و متن می​باشد. هر عنصر در منبع سند با یک برچسب شروع‌کننده شروع شده و با یک برچسب پایانی خاتمه می​یابد:

<title>Sample page</title>

عناصر همچنین می​توانند مشخصاتی برای خود داشته باشند که attribute نامیده می​شود و از اسم و مقدار تشکیل شده است:

<input name=address disabled>

مرورگران وب به عنوان برنامه​های کاربری این نشانه‌گذاری​ها را مرور کرده و آن را تبدیل به درخت DOM (Document Object Model) می​کنند. درخت DOM یک نمایش در حافظه از سند است. این درخت گره​های گوناگونی دارد. مثالی از این درخت از کد بالا را در زیر مشاهده می​کنید:



عنصر ریشه این درخت همیشه html می​باشد و عناصر دیگر با توجه به ترتیب آمدنشان در درخت قرار می​گیرند. درخت DOM می​تواند توسط scriptها مورد دسترسی قرار بگیرد و دستکاری شود. اسکریپت​ها (مثل JavaScript) برنامه​هایی هستند که در درون برچسب <script> قرار می​گیرند و یا توسط کنترل‌کننده​های رخداد در مشخصات عناصر استفاده می​شوند. برای مثال یک نوع از این اسکریپت‌نویسی را در زیر می​بینید:

<[form](http://www.w3.org/TR/2012/CR-html5-20121217/forms.html#the-form-element) [name](http://www.w3.org/TR/2012/CR-html5-20121217/forms.html#attr-form-name)="main">

Result: <[output](http://www.w3.org/TR/2012/CR-html5-20121217/forms.html#the-output-element) [name](http://www.w3.org/TR/2012/CR-html5-20121217/forms.html#attr-fe-name)="result"></output>

<[script](http://www.w3.org/TR/2012/CR-html5-20121217/scripting-1.html#the-script-element)>

[document](http://www.w3.org/TR/2012/CR-html5-20121217/dom.html#document).[forms](http://www.w3.org/TR/2012/CR-html5-20121217/dom.html#dom-document-forms).main.[elements](http://www.w3.org/TR/2012/CR-html5-20121217/forms.html#dom-form-elements).result.[value](http://www.w3.org/TR/2012/CR-html5-20121217/forms.html#dom-output-value) = 'Hello World';

</script>

</form>

هر عنصر در درخت DOM توسط یک شی نشان داده می​شود. و هر شی یک رابط کاربری برای خود دارد تا بتوان آن را دستکاری کرد. مثالی از این دستکاری:

var a = [document](http://www.w3.org/TR/2012/CR-html5-20121217/dom.html#document).[links](http://www.w3.org/TR/2012/CR-html5-20121217/dom.html#dom-document-links)[0]; // obtain the first link in the document

a.[href](http://www.w3.org/TR/2012/CR-html5-20121217/text-level-semantics.html#dom-a-href) = 'sample.html'; // change the destination URL of the link

a.[protocol](http://www.w3.org/TR/2012/CR-html5-20121217/infrastructure.html#dom-uda-protocol) = 'https'; // change just the scheme part of the URL

a.setAttribute('href', 'http://example.com/'); // change the content attribute directly

HTML در واقع یک زبان فارغ از رسانه بیان آن است، رسانه آن هرچیزی می​تواند باشد و محتوای آن می​تواند در صفحه نشان داده شود و یا خوانده شود. برای مشخص کردن نحوه نشان داده محتوای سند HTML می​توان از یک زبان استیل گذاری مانند CSS استفاده کرد. نمونه​​ای از زبان CSS را در زیر می​بینید:

background: navy; color: yellow;

HTML5 به عنوان پنجمین نسخه از زبان نشانه‌گذاری گستره جهانی وب که ابتدا تنها برای توصیف اسناد علمی​به کار می​رفت و بعدا کاربردهای گسترده تری به خود گرفت در این جهت ایجاد شد تا یک حوزه فراموش شده از وب را در بر گرفته و استانداردسازی کند. این حوزه در واقع برنامه​های تحت وب می​باشد. در نتیجه HTML5 علاوه بر بهبود مسائل قبلی در HTML4 تاکید خود را بر وارد کردن این حوزه به HTML دارد.

یکی از مهمترین عناصر اضافه شده در HTML در نسخه پنجم عنصر Canvas می​باشد که بنای طراحی و پیاده​سازی موتور فیزیک در این پروژه نیز استفاده از آن در محیط ایجاد شده توسط این عنصر به زبان JavaScript است. Canvas که لغتش به معنی بوم نقاشی است عنصری است که یک محیط و اسکریپت​هایی برای ایجاد شکل با وابستگی به رزولوشن محیط فراهم می​کند که می​تواند برای رندر کردن گراف​ها و نمودارها، گرافیک بازی​ها، شکل​های هنری و دیگر تصاویر بصری در لحظه استفاده شود. این اشکال می​توانند توسط اسکریپت​ها در Canvas به طور پویا ایجاد شوند. Canvas دو خصیصه عرض و طول دارد که در منبع سند HTML اندازه آن را در سند قابل ارائه توسط مرورگر نشان می​دهد. برای دسترسی به رابط کاربری canvas لازم است که context مربوطه را از آن دریافت کنیم.

context = canvas. [getContext](http://www.w3.org/TR/2012/CR-html5-20121217/embedded-content-0.html#dom-canvas-getcontext)(contextId [,... ])

آرگومان اول نشان دهنده نوع context مورد نظر ما و به تبع آن نوع رابط کاربری مورد نظرمان است. از جمله contextهای موجود می​توان به رابط دو بعدی و یا webgl که یک رابط سه بعدی است اشاره کرد.

2D Context در واقع رابط مورد نظر ما برای ترسیم اشکال دو بعدی در canvas است. این زمینه اشیا, متدها و مشخصه​هایی را برای ترسیم و دستکاری اشکال در سطح canvas فراهم می​کند. این زمینه روش​های گوناگونی را برای کار با اشکال ترسیمی فراهم می​کند که به این قرارند:

یک پشته برای نگهداری وضعیت ترسیم​ها که شامل ماتریس​های تبدیل، محیط بریده شده و مقدار تمام پارامترهای گرافیکی موجود از جمله:

[strokeStyle](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-strokestyle), [fillStyle](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-fillstyle), [globalAlpha](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-globalalpha), [lineWidth](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-linewidth), [lineCap](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-linecap), [lineJoin](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-linejoin), [miterLimit](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-miterlimit), [shadowOffsetX](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-shadowoffsetx), [shadowOffsetY](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-shadowoffsety), [shadowBlur](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-shadowblur),[shadowColor](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-shadowcolor), [globalCompositeOperation](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-globalcompositeoperation), [font](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-font), [textAlign](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-textalign), [textBaseline](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-textbaseline).

همچنین دو متد برای ذخیره و بازیابی این اطلاعات موجودند:

context. [save](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-save)()

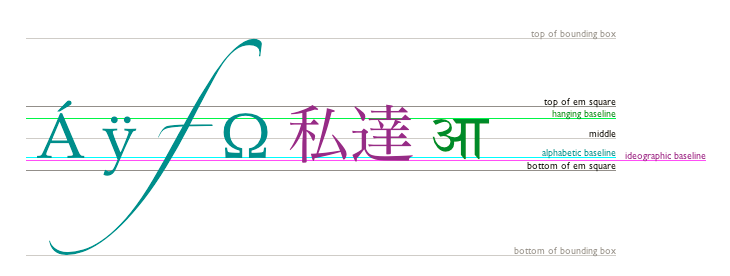
context. [restore](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-restore)()

که به مانند یک پشته عمل کرده و تمام اطلاعات را ذخیره و آخرین اطلاعات را بازیابی می​کنند.

اشیا استایل در زمینه، نوع و خصوصیات ظاهری هر عنصری را مشخص می​کنند که از آن​ها می​توان موارد زیر را نام برد:

LineStyles, Text styles, …

نمونه​​ای از انواع استایل​هایی که بر روی نوشته می​توان انجام داد را در شکل زیر مشاهده می​کنید.



شکل تنظیمات نوشته در canvas

مسیر​ها پایه ای​ترین عنصر ترسیم اشکال در زمینه هستند. هر مسیر از یک یا چند زیر مسیر تشکلی شده است و هر زیر مسیر از یک یا چند نقطه که توسط خط​های مستقیم و یا خمیده به هم متصل شده اند و همچنین یک وضعیت برای این که مشخص شود آیا مسیر بسته است یا خیر. مسیر بسته مسیری است که آخرین نقطه آن به اولین نقطه آن با یک خط مستقیم متصل می​شود. برای مثال برای ترسیم یک مسیر می​توان از کد زیر استفاده کرد:

context. [moveTo](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-moveto)(x, y)

path. [moveTo](http://www.w3.org/html/wg/drafts/2dcontext/html5_canvas/#dom-context-2d-moveto)(x, y)

و برای بستن آن:

context. closePath()

path. closePath()

برای ترسیم یک خط مستقیم تا نقطه دلخواه:

context. lineTo(x, y)

path. lineTo(x, y)

برای ترسیم یک خط خمیده Bezier تا نقطه دلخواه با نقطه کنترلی دلخواه:

context. quadraticCurveTo(cpx, cpy, x, y)

path. quadraticCurveTo(cpx, cpy, x, y)

همچنین یک خط خمیده Bezier با دو نقطه کنترلی:

context. bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

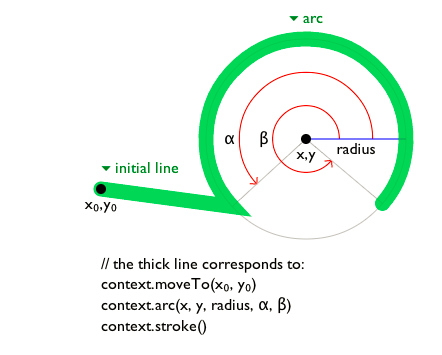
path. bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

و یک خط خمیده دایره​​ای با شعاع معلوم:

context. arcTo(x1, y1, x2, y2, radius)

path. arcTo(x1, y1, x2, y2, radius)

یک مثال از یک مسیر دوقسمتی که یک خط مستقیم و یک خط خمیده را دارد در شکل زیر مشاهده می​کنید:



شکل کشیدن خط در canvas

برای ترسیم یک مستطیل نیز می​توان از دستورات زیر بهره برد:

context. rect(x, y, w, h)

path. rect(x, y, w, h)

اشیا نوع Path برای اعلان مسیرها به کار می​روند. و توابعی برای دستکاری آنها از جمله ترکیبشان یا نوشتن کلمات دارند:

addPath(b, transform)

addPathByStrokingPath(b, styles, transform)

addText()

addPathByStrokingText()

addText()

addPathByStrokingText()

هر شی در زمینه دارای یک ماتریس تبدیل است که در ابتدای ایجاد آن شی این ماتریس یکه می​باشد. برای دستکاری این ماتریس و تبدیل اشیا متدهای زیر را در اختیار داریم:

context. scale(x, y)

context. rotate(angle)

context. translate(x, y)

context. transform(a, b, c, d, e, f)

context. setTransform(a, b, c, d, e, f)

همچنین توابع مختلفی نیز برای تغیر استایل اشیا وجود دارد:

context. fillStyle [ = value ]

context. strokeStyle [ = value ]

pattern = context. createPattern(image, repetition)

یکی دیگر از اشکال کشیدنی پایه​​ای در canvas مستطیل​ها هستند، با توابع زیر می​توان آن​ها را رسم کرد:

context. clearRect(x, y, w, h)

context. fillRect(x, y, w, h)

context. strokeRect(x, y, w, h)

همچنین برای ترسیم نوشته​ها در canvas:

context. fillText(text, x, y [, maxWidth ] )

context. strokeText(text, x, y [, maxWidth ] )

در رابطه با مسیرها بعد از مشخص شدن شکل آن​ها می​توان با توابع زیر آن​ها را ترسیم کرد:

context. beginPath()

context. fill()

context. fill(path)

context. stroke()

context. stroke(path)

context. drawSystemFocusRing(element)

context. drawSystemFocusRing(path, element)

context. scrollPathIntoView()

context. scrollPathIntoView(path)

context. clip()

context. clip(path)

context. isPointInPath(x, y)

context. isPointInPath(path, x, y)

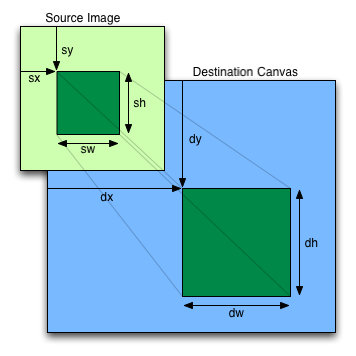
یکی دیگر از قابلیت​های اساسی canvas توانایی ترسیم عکس​ها در آن می​باشد، برای آین کار از توابع زیر می​توان استفاده کرد:

drawImage(image, dx, dy)

drawImage(image, dx, dy, dw, dh)

drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)

آرگومان​های فوق با توجه به شکل زیر تفسیر می​شوند:



شکل تصویر در canvas

در زمینه به طور کلی یکسری از نواحی به نام نواحی Hit Region یا نواحی برخورد در یک لیستی به نام hitRegions قرار می​گیرند. هر Hit Region شامل این موارد است: یک مجموعه​​ای از پیکسل​های bitmap در canvas که این ناحیه برخورد در آن قرار می​گیرد، یک محیط مرزی در bitmap در canvas که محیط ناحیه برخورد را در بر می​گیرد، و همچنین چند خصوصیت انتخابی از جمله شکل curser در آن ناحیه و غیره. با دستورات زیر این نواحی می​توانند به وجود آمده و از بین بروند:

context. addHitRegion(options)

context. removeHitRegion(options)

که در آن​ها Options می​تواند از قسمت​های زیر تشکیل شود:

path (default null)

id (default empty string)

parentID (default null)

cursor (default "inherit")

control (default null)

label (default null)

role (default null)

این نواحی برخورد کاربرد​های مختلفی می​توانند داشته باشند که از آن جمله می​توان به راحتتر کردن تشخیص برخورد​ها با ماوس اشاره کرد. همچنین این نواحی می​توانند در بازی​ها برای بست کردن به موتور فیزیک جهت تشخیص برخورد به کار بیایند.

دست آخر نیز زمینه دوبعدی ابزارهایی را هم برای دستکاری پیکسلی در اختیار می​گذارد تا چنان که ابزارهای دیگر کافی نبودند بتوان از آن​ها بهره برد تا هر شکل دلخواهی را ترسیم نمود. برای این کار باید از متدهای زیر استفاده کرد:

imagedata = context. createImageData(sw, sh)

imagedata = context. createImageData(imagedata)

imagedata = context. getImageData(sx, sy, sw, sh)

imagedata. width

imagedata. height

imagedata. data

context. putImageData(imagedata, dx, dy [, dirtyX, dirtyY, dirtyWidth, dirtyHeight ])

همچنین تمام اشکال در زمینه می​توانند دارای سایه و ضریب آلفای جداگانه​​ای باشند:

context. globalAlpha [ = value ]

context. globalCompositeOperation [ = value ]

context. shadowColor [ = value ]

context. shadowOffsetX [ = value ]

context. shadowOffsetY [ = value ]

context. shadowBlur [ = value ]

همان طور که قبلا هم اشاره شد، 2D Context زمینه اصلی مورد نظر ما در این پروژه جهت پیاده​سازی موتور فیزیک به هدف آن است، و به همین جهت توضیحات کلی در رابطه با ویژگی​ها و ابزارهای موجود برای کار با آن و همچنین رابط کاربری اش داده شد. اما زمینه دیگری که در این محیط برای canvas وجود دارد و قابل ذکر است زمینه webgl می​باشد، این زمینه یک رابط کاربری بسیار نزدیک و در رابطه با OpenGL که خود یک رابط کاربری بین زبانی و چند سکویه برای ارائه گرافیک دو بعدی و سه بعدی می​باشد ارائه می​کند. اما webgl به طور کلی رابط کاربری طراحی سه بعدی گرافیک در محیط وب بعنی canvas می​باشد. توضیح این رابط کاربری خارج از حوزه این گزارش بوده و تنها به همین اشاره اکتفا می​شود.

## جمع​بندی و نتیجه​گیری

در این فصل به بررسی کلی امکانات HTML5 و canvas و همچنین 2D Context پرداختیم و نشان دادیم که چطور این امکانات می توانند در جهت ایجاد گرافیک​های برداری در محیط یک سند وب به کار گرفته شوند. این امکانات همچنین نشان می دهند که چطور می توان یک بازی را در این محیط ایجاد کرد. آنچه در این فصل گفته شد مروری کلی بر این موضوع بود و بیشتر جهت نشان دادن بستر ایجاد موتور فیزیک مطرح شد.

در فصل آینده به بررسی ماهیت فیزیک در بازی و مطالعه انواع و شکل​های آن و همچنین توضیح این مطلب خواهیم پرداخت که در این پروژه فیزیک به چه تحوی پیاده​سازی خواهد شد.

# فصل دوم: بررسی فیزیک بازی

## مقدمه

در بازی​های ویدئویی اغلب لازم است تا شرایط و یا قسمتی از شرایط دنیای واقعی شبیه‌سازی شود، این شبیه‌سازی به این معنی است که بازیکن حس کند آنچه در بازی می​بیند می​تواند ما به ازا جسم و شی واقعی در دنیای بیرون باشد، به این صورت حس واقعی‌تر کردن بازی به هیجان و جذابیت بازی به شدت کمک خواهد کرد.

فیزیک نظام بسیار بزرگی است و فیزیک آکادمیک صدها زیر رشته دارد که هر کدام جنبه​هایی از دنیای فیزیکی را بیان می​کنند برای مثال نحوه کارکرد نور و یا یک انفجار اتمی.

اما بعضی قسمت​های فیزیک در بازی کاربرد می​تواند داشته باشد. مثلا مبحث نور شناسی برای درک حرکت نور، بازتاب​های آن بر روی جسم و در نتیجه ایجاد گرافیک​های زیبا بسیار کاربرد دارد که به آن شیوه Way-Tracing گفته می​شود اما این مبحثی از فیزیک نیست که در بازی​های کامپیوتری بتواند کاربرد داشته باشد و چون پردازشی بسیار کندی دارد در این بحث جای نمی​گیرد. یا برای مثال کاربردی از شبیه‌سازی یک نیروگاه اتمی​در بازی و فیزیک آن نمی​توان متصور بود و در نتیجه در فیزیک بازی جای نمی​گیرند.

با این بحث معلوم می​شود که هر آنچه در فیزیک هست در بازی کاربرد ندارد و در واقع وقتی صحبت از فیزیک بازی می​کنیم منظور بیشتر از همه مکانیک کلاسیک است: قوانینی که حرکات اجرام بزرگ در نتیجه نیروهایی چون جاذبه را تعیین می​کنند. در فیزیک آکادمیک این موضوعات با موضوعاتی چون کوانتوم و نسبیت جایگزین شده اند اما در دنیای بازی این قوانین می​توانند به کار گرفته شوند تا بتوان حس صلب بودن اجسام، با جرم، اینرسی، ارتجاع و شناور بودن را القا کرد.

فیزیک بازی در ابتدایی​ترین بازی​ها تا به امروز موجود بوده است و در حرکات ذرات در بازی​ها تجلی داشته است: جرقه​ها، آتش‌بازی، بالستیک گلوله​ها، دود و انفجارها. در مراحل پیشرفته‌تر فیزیک بازی برای سه دهه است که در شبیه‌سازی پرواز به کار رفته است. همچنین به فیزیک ماشین​ها با توجه به تایر، شکل، تعلیق و موتور​های مختلف اشاره کرد.

همان طور که قدرت پردازشی بالا رفت، جابجایی جعبه​های زیادی که روی هم قرار می​گیرند، دیوارهایی که تخریب می​شوند و به تکه​های خود می​شکنند نیز قابل شبیه‌سازی شدند که در حوزه فیزیک جسم صلب قرار می​گیرند. لباس​ها، پرچم​ها و طناب​ها بعدا از جمله مسائل فیزیک جسم نرم می​باشند که شبیه‌سازی شدند.

در این پروژه هدف پیاده​سازی یک موتور فیزیک ذره​​ای می​باشد که با بررسی قوانین فیزیکی آن به نحوه پیاده​سازی آن می​پردازیم.

## موتور فیزیک

در بازی​های کامپیوتری پدیده​های فیزیکی آن چنان که نیاز هستند می​توانند در خود بازی پیاده شوند و بدون نیاز به یک موتور جداگانه از پس قوانین لازم بر بیایند. اما این موضوع تنها به شرطی صحیح است که تعداد پدیده​های فیزیکی مورد نظر کم باشد، اگر تعداد پدیده​های فیزیکی که در بازی می​خواهیم به آن​ها بپردازیم زیاد باشد، پیاده​سازی آن​ها در داخل بازی غیر عملی است و لازم است که حتما آن​ها را خارج از بازی طوری پیاده کرد که پیاده​سازی بازی بتواند از توابع محاسبه‌گر پدیده​های فیزیکی‌شان استفاده کند، در این حالت یک موتور فیزیک به وجود آمده که اگر به درستی مستقل از بازی پیاده شود می​تواند برای هر بازی دیگری که نیاز به پیاده​سازی پدیده​های فیزیکی دارد به کار گرفته شود و این موضوع سبب شده است که موتورهای فیزیک عمومی به وجود بیاید که بازی​های مختلفی بتوانند از آن استفاده کنند. وجود یکی موتور فیزیک جداگانه که بتوان در یک بازی دلخواه از آن استفاده کرد سازنده بازی را از بسیاری از مسائل و پیچیدگی​ها رها کرده و فرصت بیشتری را در اختیار او قرار می​دهد، چنان که موتورهای فیزیک در طول زمان چنان قدرتمند می​شوند که پیاده​سازی جداگانه آن​ها برای هر بازی اصلا مقرون نخواهد بود.

پس می​توان این طور نتیجه گرفت که موتور فیزیک یک قطعه کد است که همه چیز در مورد فیزیک را می​داند و با هیچ وابستگی به سناریوی خاصی از بازی پیاده شده است. اساسا پدیده​های مشابه فیزیکی بسیاری هستند که می​توانند برای مثال از یک ویژگی عمومی​در یک موتور فیزیک استفاده کنند تا شبیه‌سازی شوند اما برای هر پدیده جداگانه لازم است تا مشخصات آن به موتور داده شود تا برای آن پدیده خاص شبیه‌سازی را انجام دهد.

به یک دید دیگر در این نتیجه می​توان گفت که موتور فیزیک در واقع یک ماشین حساب بزرگ و بسیار قدرتمند است که تمام محاسبات ریاضی پدیده​های فیزیکی را انجام می​دهد اما اطلاعی از این ندارد که این محاسبات دقیقا برای چه چیزی است.

لازم به ذکر است که محاسبات اشاره شده باید در نتیجه دریافت داده​هایی از طرف خود بازی باشد، که این اطلاعات منجر به محاسبات و شبیه‌سازی مورد نظر می​شود، دریافت و دادن این اطلاعات از و به موتور خارج از مبحث موتور است و به پیاده​سازی بازی مربوط می​شود.

دو فایده بزرگ در نتیجه استفاده از موتور بازی وجود دارد.

اول صرفه جویی بسیار زیاد در زمان است. یک موتور فیزیک شامل هزاران خط کد می​باشد، پیاده​سازی هر باره آن​ها برای هر بازی منطقی نمی​نماید و استفاده از یک موتور فیزیک جداگانه کاملا مقرون به صرفه است.

دوم کیفیت است. هر پدیده فیزیکی در یک موتور اختصاصی به بهترین نحو ممکن می​تواند پیاده شود و از آن​ها استفاده شود. پیاده​سازی مجدد و مجدد ان​ها حتی اگر خوب هم بشود پیاده کرد معلوم نیست که هر باز این کیفیت بتواند حفظ شود. در ضمن شاید بتوان پدیده​های مختلف را در جای خود در یک بازی پیاده​سازی کرد اما وقتی این پدیده​ها بخواهند با هم ترکیب شوند و انواع حرکات در محیط​های ترکیبی را انجام دهند آنگاه بدون وجود یک موتور فیزیک که هر کدام از پدیده​ها را جداگانه بررسی می​کند این کار عملی نخواهد بود.

## نوع و روش پیاده​سازی موتور فیزیک

برای پیاده​سازی در این پروژه از میان انواع روش​هایی که موتور فیزیک را می​توان طراحی و پیاده​سازی کرد، گزینه​های موجود را بررسی کرده و بهترین گزینه با توجه به هدف و امکانات موجود انتخاب شده است.

در اولین مسئله باید مشخص شود که موتور آیا یک موتور اجسام صلب کامل باید باشد و یا یک موتور تجمیع جرم. موتور اجسام صلب اجسام را به طور کامل و کلی در نظر گرفته و حرکات و چرخش آن​ها را پیاده می​کنند. به مانند یک جعبه که به طور کامل جسمش می​تواند شبیه‌سازی شود. موتور تجمیع جرم اجسام را به عنوان تعداد زیادی از اجرام ذره در نظر گرفته و با آن​ها به این صورت بر خورد می​کند، در واقع محاسبات برای هر ذره و در نتیجه کل جسم انجام می​شود. یک جعبه می​تواند 4 ذره باشد که در گوشه آن به وسیه میله​هایی به هم متصل شده‌اند.

موتور تجمیع اجرام از نظر پیاده​سازی دارای این مزیت است که نیازی به دانش چرخش اجسام ندارد، اما می​توان یک موتور تجمیع اجرام را با گسترش دادن به یک موتور اجسام صلب تبدیل کرد و بدین صورت از مزایای آن هم بهره‌مند شد. در این پروژه موتور پیاده​سازی شده، تجمیع اجرام است که می​تواند بعدا به اجسام صلب تبدیل شود.

در گام دوم باید نحوه تحلیل برخوردها مشخص شود. این که چگونه و با چه روشی محل برخوردها شناسایی، تحلیل و اقدام آن​ها انجام شود.

یک روش بررسی کردن و تحلیل هر برخورد به طور جداگانه و یکی یکی است. این روش، روش تکرار نامیده می​شود که مهمترین مزیت آن سرعت آن است. یک برخورد به سرعت تحلیل می​شود و تعداد زیادی برخورد در زمان کمی​به این صورت می​توانند تحلیل شوند.

یک روش دیگر که Jacoban-based نام دارد، تمام برخوردها را با هم در نظر گرفته و یک تحلیل کلی روی آن​ها انجام می​دهد. این روش بسیار زمان گیر است و بیشتر در شبیه​سازی​های فیزیکی آکادمیک به کار می​رود.

به همین دلیل در این موتور از روش اول که سریع‌تر است و مارا به نتیجه دلخواهمان می​رساند استفاده می​کنیم.

در گام بعد باید نوع ایجاد محرک حرکتی در موتور بررسی کنیم، در واقع در پیش روی ما سه راه است که تمام نیروها را همان طور که در طبیعت وجود دارند شبیه‌سازی کنیم، و یا این که آن نیروهایی که در زمان بسیار کوتاهی وارد می​شوند را به صورت تکانه[[1]](#footnote-1) و نه نیرو شبیه‌سازی کنیم و یا این که تمام نیرو​ها را در واقع به صورت تکانه​ها پیاده​سازی کنیم. در این پروژه روش دوم انتخاب شده است چرا که به حس واقعی از طبیعت بسیار نزدیک‌تر است، به این صورت نیروهایی که به مدت زیادی وارد می​شوند را به صورت همان نیرو در نظر گرفته اما نیروهایی که به مدت کمی وارد می​شوند را به صورت تکان و ضربه در نظر گرفته و این گونه آن​ها را محاسبه می​کنیم.

## جمع​بندی و نتیجه​گیری

در این فصل بررسی کردیم که موتور فیزیک در واقع چیست و در دنیای بازی​سازی به چه استفاده​ای می آید و گفتیم که این موتور در واقع به طور کلی مسئولیت شبیه​سازی پدیده​های فیزیک در دنیای واقع را دارد و از انواع مختلف آن در این پروژه به پیاده​سازی موتور تجمیع اجرام که یک موتور شبیه​سازی با کمک ذرات بدون جهت است می پردازیم.

در فصل بعد اولین قدم در را ه ساخت این موتور یعنی ایجاد فیزیک ذرات را برخواهیم داشت و با مطالعه جنبه​های مختلف یک ذره، آن را پیاده​سازی خواهیم کرد.

# فصل سوم: فیزیک ذرات

## مقدمه

موتور فیزیک ذرات در واقع موتوری است که اساس کار آن ذرات فیزیکی هستند. این ذرات دارای موقعیت، سرعت، شتاب، نیرو و غیره می باشند. بررسی این ذرات و به روز رسانی آن​ها وظیفه اصلی موتور بوده و هسته مرکزی موتور را تشکیل می دهد. در این فصل به بررسی ریاضیات آن​ها پرداخته و پیاده​سازی شان را مطرح می کنیم.

## ریاضیات ذرات

بردار یک عنصر از فضای برداری است، ساختاری که خصوصیات ریاضی خاصی برای اعمال ریاضی دارد. در این پروژه بردارهای دوبعدی در فضای معمول دوبعدی (Euclidean) مورد بحث هستند. در این حالت هر بردار نشان دهنده یک موقعیت در این فضا می​باشد. مهمترین کاربرد بردارها در مشخص کردن مختصات در فضای دو بعدی و سه بعدی می​باشد. این مختصات می​تواند توسط دو مقدار مختصاتی که هر کدام فاصله​​ای ثابت از یک محور مختصاتی هستند نشان داده می​شود. این یک سیستم مختصاتی کارتزین می​باشد.

هر بردار یک نقطه در صفحه را نشان می​دهد و هر نقطه در فضا به یک و تنها یک بردار مرتبط است. در موتور فیزیک این پروژه از آن جا که یک بردار در واقع می​تواند مشخصات بک نقطه در صفحه نیز باشد، کلاس مربوط به آن Point نامیده می​شود.

نکته مهم دیگر این است که بردار​ها همچنین می​توانند نشان دهنده جابجایی و فاصله بین نقاط صفحه باشند و درنتیجه عملیات​های جمع، ضرب و تفریق برای بردار​ها این گونه دارای معنی می​شوند.

در این پروژه عملات​های گوناگونی بر رور بردار​ها انجام می​شود که از آن جمله می​توان به نرمال کردن بردار​ها، دریافت طول آنها، زاویه آن​ها، عملیات ریاضی جمع، تقریق و ضرب​های داخلی و خارجی اشاره کرد.

در این پروژه ما تغییرات مشخصات ذرات در طول زمان را حساب می​کنیم و از آن جمله و پایه​ترین تغییر، تغییر موقعیت ذره در طول زمان بر اساس سرعت، شتاب، نیروهای وارده و غیره می​باشد. از دو دید این تغییرات را می​توان بررسی کرد. اول نحوه تغییر این متغیرها می​باشد و دوم نتیجه تغییر آن​ها. در مورد نحوه تغییر آن​ها باید به حساب دیفرانسیل رجوع کرد و نتیجه آن​ها از طریق حساب انتگرال و جمع کننده​ها میسر خواهد بود.

در حساب دیفرانسیل به این موضوع می​پردازیم که تغییرات متغیرها در طول زمان چگونه است، این تغییر سرعت است. لازم به ذکر است که سرعت دارای دو معنی velocity و speed می​باشد، که در واقع speed تنها به اندازه سرعت اشاره دارد و velocity برداری است که جهت سرعت و به عبارت دیگر تغییر را نیز مشخص می​کند. بدین ترتیب از معادله زیر می​توانیم مفهوم سرعت را به روشنی ببینیم:



که در آن p موقیعیت قبلی، p’ موقعیت جدید و v سرعت است و از آن جایی که ما به سرعت در کوتاه​ترین زمان ممکن علاقه داریم و نه سرعت متوسط در یک زمان طولانی به معادله زیر می​رسیم:



این موضوع در مورد سرعت ذره کاملا روشن است اما در مورد شتاب هم می​توان گفت که شتاب سرعت تغییر سرعت است و در نتیجه خواهیم داشت:



و اگر به نسبت مکان بخواهیم شتاب را در نظر بگیریم:



لازم به ذکر است که تمام این معادلات برای بردارها نیز صادق می​باشند چنان که هر کدام از دو مقدار آنها در این معادله​ها قرار می​گیرند و می​توان a و p را بردارهای شتاب و سرعت نامید.

حساب انتگرال در نقطه مقابل حساب دیفرانسیل قرار می​گیرد. پس ما با دانستن سرعت و موقعیت ذره در زمان خال می​توانیم موقعیت آن را در آینده پیش بینی کنیم، این یک نکته اساسی در پیاده​سازی فیزیک ذرات می​باشد. این خاصیت برای به روز کردن موقعیت و سرعت ذره در هر لحظه به کار می​رود. قطعه کدی که این کار را انجام می​دهد Integrator نام دارد.

این خاصیت انتگرال در واقع بدون وجود انتگرال​های پیچیده در ریاضیات و تنها با عملیات جبری قابل پیاده​سازی است. نحوه محاسبه سرعت و موقعیت جدید به این صورت خواهد بود:





البته موقعیت ذره در صورتی که دارای شتاب باشد از معادله زیر استخراج می​شود:



و باز هم یادآوری می​شود که این معادله​ها همگی برای برای بردارها نیز صادق می​باشند.

## جمع​بندی و نتیجه​گیری

در این فصل ذرات را به عنوان قسمت اصلی موتور بررسی کردیم و ریاضیات آن​ها را مطرح نمودیم. این ریاضیات بعدا در پیاده​سازی فصل بعد که به همراه قانون​های نیوتن مطرح می شوند به کار گرفته خواهند شد.

# فصل چهارم: قوانین حرکت

## مقدمه

موتورهای فیزیک بر پایه قوانین حرکت نیوتون بنا می​شوند. در این پروژه از این قوانین استفاده می​کنیم. سه قانون نیوتون توضیح می​دهند که موقعیت یک نقطه جرم‌دار چگونه است. این نقطه جرم‌دار را ما در این موتور فیزیک particle می​نامیم. در ادامه به بررسی پیاده​سازی آن​ها می پردازیم.

## ذره

یک ذره، موقعیت دارد ولی جهت ندارد. به مانند یک گلوله و یه ذره نور هیچ نیاز به جهت یک ذره وجود ندارد ضمن این که در فیزیک ذراتی که بررسی می​کنیم جهت ذره خارج از موضوع است. برای هر ذره باید مشخصه​های گوناگونی را نگهداری کنیم: موقعیت کنونی، سرعت، شتاب و غیره. به این صورت شیئ particle را برای ذره با prototype زیر مشخص می​کنیم.

function Particle(){

this.position = new Point();

this.velocity = new Point();

this.acceleration = new Point();

}

## دو قانون اول

در ابتدا تنها دو قانون اول را در نظر می​گیریم که به این قرار می​باشند:

یک شیئ با سرعت ثابت به حرکت خود ادامه می​دهد مگر این که نیرویی به آن وارد شود.

نیرویی که به یک شیئ وارد می​شود شتابی در آن ایجاد می​کند که با جرم آن نسبت مستقیم دارد.

قانون اول بیان می​کند که در صورت نبود هیچ نیرویی جسم چگونه عمل می​کند و به حرکت خودش با همان سرعتی که دارد ادامه می​دهد. سرعت ذره هیچگاه تغییر نمی​کند و موقعیت ان بنا به آن سرعت دائما به روز می​شود. اما در واقعیت روزمره این اتفاق نمی​افتد و ذرات پس از مقداری حرکت می​ایستند. این در واقع اثر نیروی اصطکاک و یا مقاوت هوا (drag) می​باشد. پس در واقع این نیروها به جز زمانی که در فضا حرکت کنیم همواره وجود دارند. ما نیرویdrag را در فصل تولید کننده​های نیرو بررسی خواهیم کرد اما به جز این نیرو می​توانیم از یک نوع خاص نیروی مقاومت محیط در خود شئی particle بهره برد که در مواقع لازم می​توان از آن استفاده کرد بدون این که یک نیروی اضافه لازم باشد در فیزیک تعریف کنیم. این مقاوت را damping می​نامیم.

this.damping = null;

به این صورت با توجه به مقدار damping یک نسبتی از سرعت شیئ را در هر به روز رسانی در integrator کم می​کنیم. اگر damping مقدار صفر را داشته باشد، بعد از update هیچ مقداری از سرعت باقی نخواهد ماند و مقدار یک به معنی این است که ذره هیچ مقدار از سرعتش را از دست نخواهد داد. به این صورت مقادیر نزدیک به یک برای شبیه‌سازی مقاومت محیط نزدیک به واقعیت هستند.

قانون دوم مکانیزم تاثیر نیرو را به ما می​دهد، به این صورت که نیرو چیزی است که شتاب ذره را تغییر می​دهد، بنابراین ما نمی​توانیم کاری کنیم که موقعیت یا سرعت جسم مستیما تغییر کند، و باید با اعمال نیرو شتاب آن را تغییر دهیم تا به سرعت و یا موقعیت مورد نظر برسد. بنابراین برای این که پرش و یا اتفاقات غیر عادی فیزیک ذره نیفتد برای ایجاد تغییر در ذره موقعیت و سرعت آن را مستقیما تغییر نمی​دهیم بلکه شتاب آن را تغییر می​دهیم تا حرکتش حالت طبیعی داشته باشد.

بنابراین معادله دوم تعیین می​کند که چگونه نیرو به شتاب ربط پیدا می​کند. معادله نیرو و ارتباط آن به شتاب به این صورت می​باشد:



و درنتیجه برای بدیت آوردن شتاب ناشی از نیروی وارده داریم:



به طوری که f نیرو و m جرم ذره می​باشد. و البته نیرو هم به مانند شتاب در این جا بردار است.

با توجه به این توضیحات ضروری می​آید که جرم ذره را نیز به عنوان یک متغیر به سیئ particle اضافه کنیم. اما این کار را به صورت مستقیم انجام نمی​دهیم. به دلیل این که یک ذره هیچگاه نمی​تواند جرم صفر داشته باشد و وجود داشتن جرم صفر با توجه به معادله اخیر منجر به خطای تقسیم بر صفر می​شود و در واقع به معنی شتاب بی نهایت است و همچنین به این دلیل که جرم بی نهایت در موتور فیزیک می​تواند دارای معنی باشد به این صورت که ذره جرم بسیار بزرگی دارد که حرکت نمی​کند و هیچ محاسبات حرکتی برای آن لازم نیست، به مانند گیره​هایی برای پاندول و یا دیوارها و مرزهای محیطی و در زبان JavaScript نمی​توانیم یک عدد بی نهایت نشان دهیم پس به جای خود جرم جسم، معکوس جرم جسم را در particle قرار می​دهیم، به این صورت هم امکان به وجود داشتن ذره با جرم صفر را از بین برده‌ایم و هم این که با صفر قرار دادن معکوس جرم توانسته‌ایم جرم بی نهایت را پیاده​سازی کنیم:

this.inverseMass = null;

جاذبه یکی از مهمترین نیروهای موجود در طبیعت می​باشد و بین هر دو جسم وجود دارد که به جرم آن​ها و فاصله شان مربوط است. قانون جهانی گرانش که توسط نیوتون بیان شده است دارای معادله زیر است:



که m جرم جسم​ها، r فاصله آن​ها و G ثابت جهانی گرانش است. این جاذبه بین اجسام کوچک قابل صرف نظر است و بین ما و زمین بسیار زیاد می​باشد بنابراین چون ما تنها برای جاذبه زمین می​خواهیم در موتور جایی قرار دهیم از این معادله صرف نظر کرده و به معادله ساده‌تر زیر برای هر جسم در زمین می​رسیم: 

که در آن g شتاب گرانش زمین است و از معادله قبلی به دست می​آید:



بنابراین فارق از جرم جسم شتاب گرانش همواره ثابت خواهد بود. ما دو راه بنابراین برای اعمال جاذبه به ذرات در این موتور فیزیک داریم یکی اعمال مستقیم شتاب جاذبه به آن​ها و دیگری ایجاد یک تولید کننده نیرو در موتور که در فصل خود مورد توضیح قرار می​گیرد اما علیرغم این که در آخر در این موتور راه دوم در پیش گرفته خواهد شد اما فعلا تا زمان ایجاد تولید کننده​های نیرو در مثال​ها از روش اول استفاده می​کنیم. مقدار g حتی می​تواند برای هر ذره متفاوت باشد و به صورت برداری که مقدار y آن –g نشان داده شود.

## اعمال معادلات

بنابر تمام توضیحاتی که تا کنون داده شد می​توان حالا Integrator را برای particle پیاده​سازی کرد. در هر فریم تصویر integrator شتاب هر ذره را بررسی می​کند و عمل به روز رسانی​ها را با توجه به آن انجام می​دهد. در این جا integrator دارای دو قسمت است یکی به روز کردن موقعیت ذره با توجه به سرعتش و دیگری به روز کردن سرعت آن با توجه به شتابش. Integrator همچنین نیازمند یک پارامتر زمان است تا این به روز رسانی​ها را بتواند انجام دهد. برای این کار مدت زمان سپری شده بین هر فریم را به آن داده تا محاسبات لازیم بین هر فریم را انجام دهد.

معادله به روز رسانی موقیعت ذره چنان که قبلا مشاهده کردیم به صورت زیر است:



اما قسمت سوم معادله چنان که زمان بین هر فریم بسیار کم می​باشد و چنانچه در خودش ضرب شود بسیار عدد کوچکی خواهد بود و در محاسبات تاثیری نخواهد گذاشت را می​توان حذف کرد و به معادله زیر رسید:



و همچنین چنان که دیدیم معدله به روز رسانی سرعت ذره نیز به این صورت خواهد بود:



اما چنان که قبلا هم اشاره شد ما از damping برای کم کردن مقداری از سرعت در هر به روز رسانی استفاده می​کنیم. این کار با معادله زیر امکان پذیر است:



که در آن d همان مقدار damping است که با به توان زمان رساندن آن فاکتور زمان را هم در آن دخیل کرده‌ایم.

به این صورت می​توانیم Integrator را به صورت زیر پیاده کنیم:

Particle.prototype.integrate = function(duration){

if(duration < 0){

//exception

console.log("error in duration");

}

this.position.addScaledVector(this.velocity, duration);

var resultingAcc = new Point(this.acceleration.x,this.acceleration.y);

resultingAcc.addScaledVector(this.forceAccum, this.inverseMass);

this.velocity.addScaledVector(resultingAcc, duration);

this.velocity.multiplebyScalar(Math.pow(this.damping, duration));

if(this.bindedShape!=null){

this.bindedShape.position = this.position;

}

this.forceAccum.x=0;

this.forceAccum.y=0;

this.path.segments[1].point = this.position;

}

## جمع​بندی و نتیجه​گیری

در این فصل قوانین حرکت را بررسی کردیم و دیدیم که چگونه می توان موقعیت، سرعت و شتاب را در یک کلاس برا ذرات پیاده​سازی کرد. این پیاده​سازی​ها در آخر برای به روز رسانی به Integrator نیاز خواهد داشت که معادلات حرکت را اعمال کند.

در فصل بعد به چند نمونه استفاده از این پیاده​سازی خواهیم پرداخت.

# فصل پنجم: استفاده از فیزیک ذره​​ای در موتور

## مقدمه

با مبانی فیزیکی که تا همین جا بحث شده است می​توان اولین شبیه​سازی​های تحت موتور را به وجود آورد که شبیه‌سازی حرکات تحت تاثیر شتاب و سرعت اولیه می​باشد. نکته اساسی در این است که با این که تا این جا کد بسیار زیادی تولید نشده است اما تئوری فیزیک پشت آن بسیار غنی و مهم می​باشد. تا این جای کار موتور فیزیک محدودیت​های بسیاری دارد که در فصل​های به آن​ها خواهیم پرداخت برای مثال ذرات تا این جا شیئ​هایی ایزوله هستند که با محیط خود تعاملی ندارند اما بعدا این تعامل باید ایجاد شود. در این فصل از موتور برای پردازش پرتابه​ها استفاده می​کنیم. این پرتابه​ها شامل گلوله، توپ و غیره می​شود. همچنین از موتور برای ایجاد آتش‌بازی استفاده خواهیم کرد.

## پرتابه​ها

یکی از بیشترین کاربردهای موتور فیزیک شبیه‌سازی پرتابه​ها می​باشد. در مثال این پروژه هر اسلحه یک نوع ذره را شلیک می​کند. این ذرات که آن​ها را پرتابه می​نامیم شامل انواع مختلفی می​تواند باشد از جمله گلوله تفنگ، توپ جنگی و یا گلوله​های انفجاری.

هر اسلحه مشخصات دهانه خاص خود را دارد که در نتیجه آن سرعت اولیه خاص خد را به پرتابه می​دهد که برای مثال برای یک اسلحه لیزری بسیار سریع و برای یک توپ آتشین کندتر است.

باید توجه داشت که برای انتخاب سرعت اولیه برای پرتابه​ها آن چه در واقع و در دنیای بیرونی وجود دارد احتمالا در موتور فیزیک کاربرد نخواهد داشت و باید دستکاری شود. برای مثال برای شبیه‌سازی لیزر منطقی نخواهد بود که سرعت آن را 300،000.000 متر در ثانیه در نظر بگیریم چرا که در این صورت، اثری از آن در بازی دیده نخواهد شد و اگر بخواهیم تنها اثر آن را در اجسام دیگر ببینیم منطقی نخواهد بود که اصلا پرتابش کنیم، بلکه می​توانیم نقطه اثر را مستقیما پیدا کنیم. اما چون در این مثال خود پرتاب شدن لیزر را می​خواهیم مشاهده کنیم می​توانیم با سرعت کمتری تا جایی که دیده شود اما بالا بودن سرعت آن نیز احساس شود پرتابه مربوط به لیزر را پرتاب کنیم. در ضمن احتمالا لازم است که جرم جسم نیز بیشتر از آنچه در عالم واقع هست در نظر گرفته شود. این موضوع سبب می​شود که انرژی آن​ها در هنگام برخورد افزایش پیدا کند و اثر تخریبی بیشتری داشته باشد. در ضمن در مورد شتاب جاذبه نیز باید دستکاری​هایی انجام دهیم تا برای هر پرتابه سرعت پایین رفتن آن متناسب با نوع پرتابه باشد. یک راه ساده برای تعیین اندازه جاذبه برای هر پرتابه انتخاب آن با توجه به سرعت آن است به این صورت که برای هر تغییر در سرعت پرتابه جاذبه می​تواند از معادله زیر بدست آید:



که در آن g نرمال شتابی است که می​خواهیم شبیه‌سازی کنیم و در دنیای واقع 10 متر بر مجذور ثانیه می​باشد. در نتیجه برای پرتابه​​ای مثل گلوله این مقدار به 0.5 متر بر مجذور ثانیه خواهد رسید.

در کد زیر پیاده​سازی چهار پرتابه را با کمک این موتور فیزیک که تا این جا طراحی کرده‌ایم آورده شده است. در این کد چهار پرتابه شبیه‌سازی شده اند که گلوله تفنگ، توپ جنگی، توپ آتشین (که بعد از پرتاب آتش گرفته و در صورت عدم برخورد بر اثر جرم کم به هوا می​رود) و لیزر هستند. برای هر پرتابه سرعت اولیه، شتاب اولیه، مقدار damping، و مقدار جرم آن را مشخص کرده‌ایم. مکان اولیه همه آن​ها یکسان ور در سمت چپ صفحه است. کاربر با زدن هر عدد از یک تا چهار یکی از آن​ها را می​تواند پرتاب کند:

var ball;

tool.onKeyDown = function(event){

if(event.character=='1'){

var ball1 = new Particle();

ball1.position = initialPosition;

ball1.inverseMass = 1/2;//kg

ball1.velocity = new Point(350,0);

ball1.acceleration = new Point(0,10);

ball1.damping = 0.99;

ball1.forceAccum = new Point(0,0);

ball1.bindedShape = myCircle;

ball = ball1;

}else if(event.character=='2'){

var ball2 = new Particle();

ball2.position = initialPosition;

ball2.inverseMass = 1/200;//kg

ball2.velocity = new Point(400,-300);

ball2.acceleration = new Point(0,200);

ball2.damping = 0.99;

ball2.forceAccum = new Point(0,0);

ball2.bindedShape = myCircle;

ball = ball2;

}else if(event.character=='3'){

var ball3 = new Particle();

ball3.position = initialPosition;

ball3.inverseMass = 1;//kg

ball3.velocity = new Point(250,0);

ball3.acceleration = new Point(0,-30);

ball3.damping = 0.9;

ball3.forceAccum = new Point(0,0);

ball3.bindedShape = myCircle;

ball = ball3;

}else if(event.character=='4'){

var ball4 = new Particle();

ball4.position = initialPosition;

ball4.inverseMass = 10;//kg

ball4.velocity = new Point(1000,0);

ball4.acceleration = new Point(0,0);

ball4.damping = 0.99;

ball4.forceAccum = new Point(0,0);

ball4.bindedShape = myCircle;

ball = ball4;

}

}

همچنین در مورد به روز رسانی فیزیک هم لازم است کد زیر توجه شود که در هر فریم مدت زمان گذشت آن فریم گرفته می​شود و به Integrator پرتابه داده می​شود تا محاسبات لازم را انجام دهد و پرتابه را به روز کند:

onFrame = function(event){

if(ball){

ball.integrate(event.delta);

}

}

## آتش‌بازی

مثال آتش‌بازی نیز می​تواند توسط موتور فیزیکی که تا این جا طراحی کرده‌ایم مورد پیاده​سازی قرار بگیرد کاربردهایی که این پیاده​سازی می​تواند داشته باشد علاوه بر خود آتش‌بازی انواع انفجارها، پخش شدن آب و حتی دود و آتش است.

برای ایجاد آتش‌بازی باید به نوع Particle چند داده دیگر را نیز اضافه کنیم تا تبدیل به یک شیئ انفجاری شود. اشیا انفجاری اول از همه حاوی بار انفجاری هستند که آن​ها را payload می​نامیم. راکت اولیه که پرتاب می​شود به چندین شیئ انفجاری کوچک دیگر تبدیل می​شود که آن​ها هم خود بعد از انفجار می​توانند بار انفجاری داشته باشند. به این صورت اشیا انفجاری نوع​های مختلفی دارند که آن​ها را باید در برنامه در نظر بگیریم.

نکته دوم در آتش‌بازی عمر هر ذره است. باید برای هر ذره انفجاری عمری تعیین شود تا به کمک آن مشخص شود که بعد از مدتی آن ذره منفجر شده و از بین می​رود، در هنگام انفجار آن نیز بار انفجاری تخلیه می​شود. این عمر در هر به روز رسانی در هر فریم باید کم شود.

بنابراین برای داشتن این اطلاعات اضافه ما نوع particle را با ارث‌بری به نوع firework گسترش می​دهیم:

function Firework(){

Particle.call(this);

this.type = null;

this.age = null;

}

Firework.prototype = new Particle();

Firework.prototype.constructor = Firework;

Firework.prototype.update = function(duration)

{

this.integrate(duration);

this.age -= duration;

return (this.age < 0);

}

این موضوع که هر نوعی از شیئ انفجاری چه خصوصیاتی دارد و مثلا بار انفجاری دارد و اگر دارد آن باز​ها چقدر و هر کدام از چه نوعی هستند را نیز باید با ساختاری مناسب در کد بگنجانیم برای این کار ساختار FireworkRule را در نظر گرفته‌ایم تمام این اطلاعات را در بر می​گیرد.

function FireworkRule()

{

this.type = null;

this.minAge = null;

this.maxAge = null;

this.minVelocity = null;//vector

this.maxVelocity = null;//vector

this.damping = null;

this.payloadCount = null;

this.payloads = new Array();

}

function Payload(type,count)

{

this.type = type;

this.count = count;

}

قوانین در کد تعریف می​شوند و لیستی از آن​ها به وجود می​آید و لازم به توضیح جزئیات آن​ها در این جا نمی​باشد.

در پیاده​سازی هر شیئ انفجاری عمرش به روز می​شود و با قوانین مربوط به خود مقایسه می​شود. اگر عمرش تمام شود از ازی حذف شده و اشیا انفجاری دیگری به جایش با توجه به قانون متناظرش ایجاد می​شود. کدی که اشیا انفجاری جدید را می​سازد به این صورت است:

FireworkRule.prototype.create = function(parent)

{

var firework = new Firework();

firework.type = this.type;

firework.age = (this.maxAge-this.minAge)\*Math.random()+this.minAge;

if (parent!=null){

var position = new Point();

position.x = parent.position.x;

position.y = parent.position.y;

firework.position = position;

var vel = new Point();

vel.x = parent.velocity.x;

vel.y = parent.velocity.y;

vel.x += (this.maxVelocity.x-this.minVelocity.x)\*Math.random()+this.minVelocity.x;

vel.y += (this.maxVelocity.y-this.minVelocity.y)\*Math.random()+this.minVelocity.y;

firework.velocity = vel;

} else{

firework.position = new Point(300\*Math.random()+200,650);

var vel2 = new Point(0,0);

vel2.x += (this.maxVelocity.x-this.minVelocity.x)\*Math.random()+this.minVelocity.x;

vel2.y += (this.maxVelocity.y-this.minVelocity.y)\*Math.random()+this.minVelocity.y;

firework.velocity = vel2;

}

firework.inverseMass = 1;

firework.damping = this.damping;

firework.acceleration = new Point(0,50);

return firework;

{

وقتی اشیا ایجاد می​شوند مقدار متغیرهایشان با توجه به متغیرهای شیئی که در ساختشان نقش داشته و قوانین مربوط به خودشان مقداردهی می​شود. Damping آن​ها با توجه به میزان مسافتی که لازم است بپیمایند تعیین می​شود و جاذبه نیز بر آن​ها تاثیرگذار است.

در هر فریم تمام اشیا موجود به روز می​شوند و اگر هر کدام تمام شود اقدام لازم انجام می​شود.

for(var i =0 ; i< fireworks.length;i++){

var firework = fireworks[i];

if (firework.type > 0)

{

if (firework.update(event.delta))

{

var rule = rules[firework.type-1];

firework.type = 0;

firework.bindedShape.remove();

var index = fireworks.indexOf(firework);

fireworks.splice(index, 1);

for (var j = 0; j < rule.payloadCount; j++)

{

var payload = rule.payloads[j];

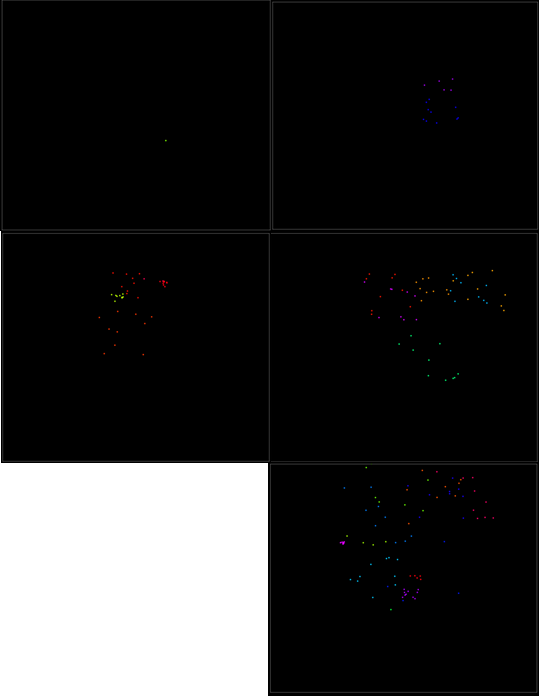
create(payload.type, payload.count, firework);

}

}

}

در شکل زیر نمونه​هایی از لحظاتی از اجرای این مثال آورده شده است.



شکل مثال آتش بازی

## جمع​بندی و نتیجه​گیری

بنابراین همان طور که مشاهده کردیم موتور فیزیکی که تا این جا طراحی و ایجاد کرده‌ایم می​تواند برای ایجاد جلوه​های ویژه به کار رود. حرکت پرتابی پرتابه​ها، سیتم​های ذره​ها و جلوه​های ویژه انفجار​ها از این جمله هستند. با انجام تنظیمات فراهم شده برای ذرات چون جاذبه، مقاومت محیط و سرعت اولیه می​توان اکنون انواع پدیده​های فیزیکی چون حتی آبا سیال و دود و آتش‌بازی را شبیه‌سازی کرد. در قسمت​های بعد گسترش این موتور فیزیک را بررسی می​کنیم تا انواع دیگر پدیده​های فیزیکی و اشیا را بتوانیم شبیه‌سازی کنیم.

# فصل ششم: فیزیک تجمیع اجرام، نیروهای عمومی

## مقدمه

در طراحی موتور فیزیک تا این جا نیروهایی چون جاذبه زمین را با مشخص کردن شتاب آن​ها در شبیه​سازی گنجانده‌ایم اما لازم است تا موتور را برای این که با هر تعداد نیرو که به هر تعداد ذره وارد می​شود عمل کند گسترش دهیم. جاذبه را نیرویی جداگانه در نظر بگیریم و تولید کننده​های نیرو را بررسی کنیم.

## اصل دالامبر[[2]](#footnote-2)

ما معادله​های لازم برای این که رفتار شیئ تحت تاثیر یک نیرو را تحلیل کنیم در اختیار داریم اما اگر چند نیرو همزمان بر یک شیئ تاثیر بگذارند چگونه باید رفتار شیئ را مشخص کنیم؟ نیروهای مختلفی که بر روی ذره تاثیر می​گذارن هر کدام اثر متفاوتی بر روی آن خواهند گذاشت و جمع این آثار برای تحلیل در موتور بسیار اهمیت دارد. اما اگر بتوان تاثیر تمام نیروها را یکجا بررسی کرد کمک بزرگی در پیاده​سازی موتور خواهد بود.

اصل دالامبر در این جا به کمک می​آید. هر چند تمام اصل در واقع پیچیده و برای موتور بدون کاربرد است و در واقع کمیت​های در ایجاد معادله​های حرکت را به هم مربوط می​کند. اما یک کاربرد بسیار مهم از این اصل را می​توان در این جا بیان و از آن استفاده کرد.

این کاربرد به این قرار است که اگر یک مجموعه از نیروها بر روی یک شیئ تاثیر بگذارند می​توان تمام آن نیروها را با یک نیرو جایگزین کرد که به این صورت محاسبه می​شود:



به عبارت دیگر به سادگی می​توانیم تمام نیروهای وارده را با جمع برداری با هم جمع کرده و یک نیروی واحد بدست آوریم و سپس اثر آن را به عنوان اثر کل نیروها به کار بگیریم.

برای این کار یک بردار جمع نیروها در particle در نظر می​گیریم که در هر بار به روز رسانی آن را صفر کرده و نیروهای وارده را سپس در آن جمع می​کنیم. نیروی جمع شده آخر را سپس در محاسبات وارد می​کنیم:

this.forceAccum = new Point(0,0);

برای این که نیرو​ها را به ذرات بتوانیم وارد کنیم می​شود آن​ها را در هر بار به روز رسانی در forceAccume جمع کرد تا در ذره اثر بگذارند امااین تنها برای چند نیروی محدود و مدت زمان محدود امکان پذیر است. باید روشی را اتخاذ کنیم تا برای هر چند نیرو و برای هر چند ذره و هر چقدر زمان بتوانیم این کار را انجام دهیم. برای همین از یک registry برای ثبت نیروها و ذره اثر پذیرشان استفاده می​کنیم. در یک رجیستری هر نیرو با یک ذره ثبت می​شود که باید برای آن در هر فریم نیرو را تامین کند. این مولد​های نیرو را در برنامه force generator می​نامیم.

## تامین‌کنندگان نیرو

در این موتور فیزیک مکانیزم تاثیر چندین نیرو با هم را ایجاد کرده‌ایم اما این که نیروها چگونه و از کجا به وجود می​آیند را نیز باید دقیقا مشخص کنیم. نیرویی مانند جاذبه در این مورد ساده به نظر می​رسند، چون همیشه برای همه ذرات وجود دارد. اما نیروهای دیگر لزوما به این شکل نیستند.

بعضی نیروها به دلیل رفتار شیئ به وجود می​آیند. برای مثال نیروی مقاومت محیط برای هر ذره با سرعت​های متفاوت متفاوت است. نیروهایی کاملا از محیطی می​آیند که ذره در آن​ها هست. مثل نیروی رانش آب به ذره شناور. بعضی نیروها در ارتباط بین ذرات با هم به وجود می​آیند مانند نیروی ناشی از فنر که در فصل بعد بررسی می​شود. و در آخر نیروهای دیگری هستند که از درخواست کاربر چه حقیقی و چه کنترل شده توسط هوش مصنوعی به وجود می​آیند به مانند درخواست افزایش شتاب ماشین.

پیچیدگی دیگر نیروها در طبیعت پویای بعضی از آن​هاست. نیرویی مثل نیروی جاذبه زمین از نظر ساده است چون همیشه و همه جا ثابت است. می​توان آن را تنها یک بار محاسبه کرد و در تمام بازی به کارش برد. اما اکثر نیروهای دیگر دائما در حال تغییرند. بعضی با توجه به تغییر موقعیت ذزه و یا سرعت آن تغییر می​کنند: نیروی مقاومت محیط در سرعت​های بالاتر بیشتر است و نیروی ناشی از فنر در فشرگی​های بیشتر، باز بیشتر است. بعضی دیگر اما با توجه به عوامل خارجی تغییر می​کنند: یک انفجار یا تمام شدن سوخت یک محرک.

ما باید قادر باشیم تا با انواع مختلفی از نیروها که با انواع متفاوتی از مکانیزم​ها برای محاسبات شان وجود دارند کار کنیم و آن را در موتور خود داشته باشیم. بعضی ثابت‌اند، بعضی توابعی را روی برخی خصوصیات ذره​ها اعمال می​کنند، بعضی ورودی از کاربر می​خواهند و بعضی وابسته به زمانند. اگر تمام این انواع را بخواهیم در کد پیاده​سازی کنیم و از کاربر انتظار داشته باشیم که برای انواع نیروهایی که در نظر دارد آن​ها را با هم ترکیب و از بینشان انتخاب کند بعد از چند نوع پیاده​سازی به کدی خواهیم رسید که از نظر مدیریتی بسیار سخت و پیچیده خواهد شد. به طور ایده‌آل اما در نظر داریم که نحوه محاسبه و تولید نیرو را بتوانیم از بقیه قسمت​ها ایزوله کنیم و به موتور این اجازه را بدهیم که فارغ از این که نیرو چگونه تولید می​شود با آن بتواند کار کند. به این صورت می​توانیم هر تعداد نیرو که بخواهیم به یک شیئ وارد کنیم بدون این که نیاز باشد که شیئ از نحوه تولید نیروها با خبر شود.

این کار با ساختاری به نام force generator در این موتور امکان پذیر است. در واقع به تعداد انواع نیروها می​تواند مولد نیرو وجود داشته باشد همان شیئ از درون آن​ها و نوع آن​ها نیاز ندارد که باخبر باشد. ذرات از یک رابط ثابت برای یافتن نیرویی که به آن​ها وارد می​شود استفاده می​کنند و این نیروها در قسمت integrator با هم جمع شده و اعمال می​شوند. به این صورت می​توانیم هر تعداد نیرو با هر نوعی را به هر ذره​​ای وارد کنیم. و درضمن می​توانیم برای هر بازی و یا هر مرحله این نوع​های جدیدی از نیرو را به راحتی ایجاد کنیم بدون این که نیاز باشد تا تغییرات عمده​​ای در موتور انجام دهیم و کدی را بازنویسی کنیم.

برای رسیدن به این مقصود نیاز داریم تا از یک مفهوم شیئ‌گرایی در JavaScript استفاده کنیم که در prototype-based بودن این زبان نهفته است و آن توانایی بسیار بالای آن در ایجاد polymorphism و استفاده از pseudo-interface​ها می​باشد. در زبان​های شیئ‌گرا interface در واقع مشخصات کارکردی یک نوع را بیان می​کند و مشخص می​کند که یک نوع شیئ چگونه با نوع​های دیگر ارتباط برقرار می​کند و در واقع توابعی از آن نوع که دیگران می​توانند استفاده کنند را در معرض می​گذارد. شیئ​هایی که این قرارداد را رعایت کنند درواقع آن interface را پیاده کرده‌اند. قدرت interfaceها در واقع در polymorphism نهفته است که توانایی یک زبان برای استفاده از قطعه​های نرم​افزاری است که آن interface را پیاده کرده‌اند. در واقع کد استفاده کننده از یک interface می​تواند از هر شیئی که آن را پیاده کرده باشد با توجه به توابهی که در آن توضیح داده شده است استفاده کند. کد صدا زننده هرگز از پیاده​سازی درونی خبر ندارد چنان که برای هر شیئ هم متفاوت است. جانشین‌پذیری در واقع نکته اساسی این خاصیت است چنان که ما یک interface برای مولد​های نیرو داریم که می​توانیم از آن​ها برای تامین نیرو استفاده کنیم، حال هر نوع مولد نیرو را می​توانیم ایجاد کنیم و ذره​ها بدون دانستن نوع آن​ها می​توانند از آن​ها استفاده کنند.

در پیاده​سازی مولد​های نیرو، آن​ها تنها نیروی زمان حال را ارائه می​کنند که در نیروهای دیگر جمع شده و به ذره اعمال می​شود:

function ParticleForceGenerator(){

}

ParticleForceGenerator.prototype.updateForce = function(particle,duration){

}

متد updateForce با زمان فریمی​که نیرو را در آن می​خواهیم و خود ذره فراخوانی می​شود. این زمان در برخی از تامین کنندگان نیرو ضروری است، مانند تامین‌کنندگان نیروی فنری که به شدت به این زمان وابسته‌اند.

در ضمن خود particle را هر بار به این متد می​دهیم تا نیاز نباشد خود مولد نیرو از مشخصات آن جداگانه نگهداری کند. به این صورت یک نمونه از یک مولد می​تواند برای چندین ذره به کار رود و برای هر کدام که تابعش فراخوانی شود کار محاسبات را انجام دهد. همچنین مولد نیرو مقداری را باز نمی​گرداند، به این صورت انعطاف‌پذیری آن حفظ شده و برای به روز کردن نیروی ذره مستقیما خود عمل می​کند تا انواع عملیت دلخواه را بتواند انجام دهد به جای این که مقداری را بازگرداند.

همچنین لازم است که ساختارهایی را برای حفظ تامین‌کنندگان و ذرات نظیرشان در نظر بگیریم. برای رسیدن به بهترین کارایی اجرایی و حافظه یک ساختار ثبت کلی و مرکزی برای همه ذرات و تامین کنندگان به وجود می​آوریم.

function ParticleForceRegistry(){

this.registrations = new Array();

}

function ParticleForceRegistration(){

this.particle = null;

this.particleForceGenerator = null;

}

در هر فریم قبل از این که به روز رسانی انجام شود تمام تامین کنندگان نیرو صدا زده می​شوند و نیروهای خود را به ذرات اضافه می​کنند تا بعدا در به روز رسانی به شتاب مناسب برسند:

ParticleForceRegistry.prototype.updateForces = function(duration){

for(var i=0;i<this.registrations.length;i++){

this.registrations[i].particleForceGenerator.updateForce(this.registrations[i].particle, duration)

}

}

با توجه به پیاده​سازی ساختاری که تاکنون داشته‌ایم اکنون می​توانیم یک مولد نیرو برای جاذبه زمین ایجاد کنیم و به جای روش قبلی که یک شتاب ثابت را به ذره وارد می​کرد قرار دهیم. این پیاده​سازی به این شکل خواهد بود:

function ParticleGravity(){

ParticleForceGenerator.call(this);

this.gravity = new Point(0,10);

}

ParticleGravity.prototype = new ParticleForceGenerator();

ParticleGravity.prototype.constructor = ParticleGravity;

ParticleGravity.prototype.updateForce = function(particle,duration){

if(!particle.infiniteMass){

particle.addForce(new Point((1/particle.inverseMass)\*this.gravity.x,(1/particle.inverseMass)\*this.gravity.y));

}

}

نیرو با نجوه به جرم جسمی​که به تابع داده شده است محاسبه می​شود، تنها داده​​ای که توسط اسن شیئ نگهداری می​شود شتاب گرانش است و یک نمونه از این شیئ می​تواند به هر تعداد ذره اعمال شود.

اکنون می​توانیم نیروهای دیگری را نیز پیاده​سازی کنیم. نیروی بعدی، نیروی مقاومت محیطی است. این نیرو، نیرویی است که به بدنه شیئ وارد می​شود و به سرعت آن وابستگی دارد. معادلات نیروی مقاومت محیطی معادلات پیچیده​​ای هستند که در زمان واقعی قابل پیاده​سازی نمی​باشند. اما در موتور فیزیک یک نوع ساده‌تر از این معادلات را به کار می​بریم که از معادله زیر بدست می​آید:



که در آنها دو ضریب k ثابت​هایی هستند که مشخص می​کنند که نیروی مقاومت چقدر قوی است. این ثابت​ها ضریب​های مقاومت محیطی نامیده می​شوند که هم به نوع شیئ و هم نوع محیط بستگی دارند.

این معادله بیان می​کند که این نیرو در جهت مخالف سرعت ذره وارد می​شود با قدرتی که به مقدار سرعت و مجذور این مقدار رابطه مستقیم دارد.

نیروی مقاومتی که ضریب دوم را دارد سرعت بسیار بالاتری رشد می​کند. این همان دلیلی است که یک خودرو نمی​تواند از یک سرعت بیشتر حرکت کند والا به سرعت بی نهایت هم می​رسید. در سرعت​های کم خودرو عملا هیچ نیرویی را از محیط حس نمی​کند ولی برای هر دو برابر شدن سرعتش، این نیرو چهار برابر می​شود.

پیاده​سازی این نیرو به شکل زیر می​باشد:

function ParticleDrag(){

ParticleForceGenerator.call(this);

this.k1 = 1;

this.k2 = 0;

}

ParticleDrag.prototype = new ParticleForceGenerator();

ParticleDrag.prototype.constructor = ParticleDrag;

ParticleDrag.prototype.updateForce = function(particle,duration){

var force;

force = particle.velocity;

var dragCoeff = force.length;

dragCoeff = this.k1 \* dragCoeff + this.k2 \* dragCoeff \* dragCoeff;

force = force.normalize();

force.x \*= -dragCoeff;

force.y \*= -dragCoeff;

particle.addForce(force);

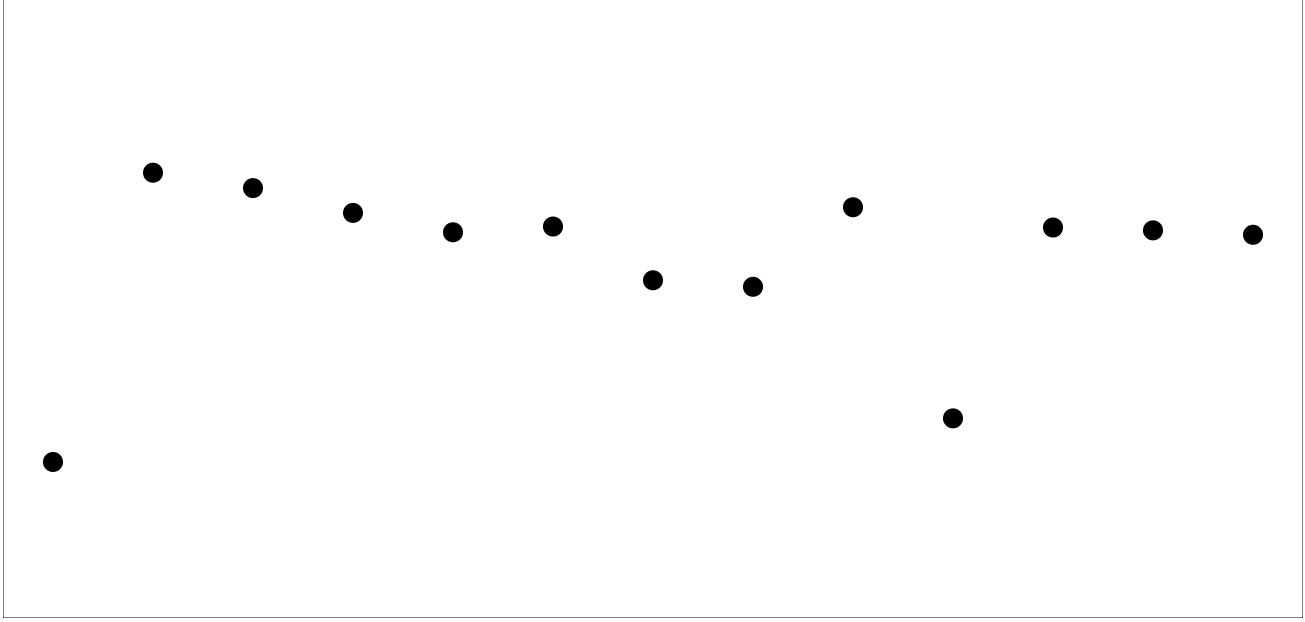
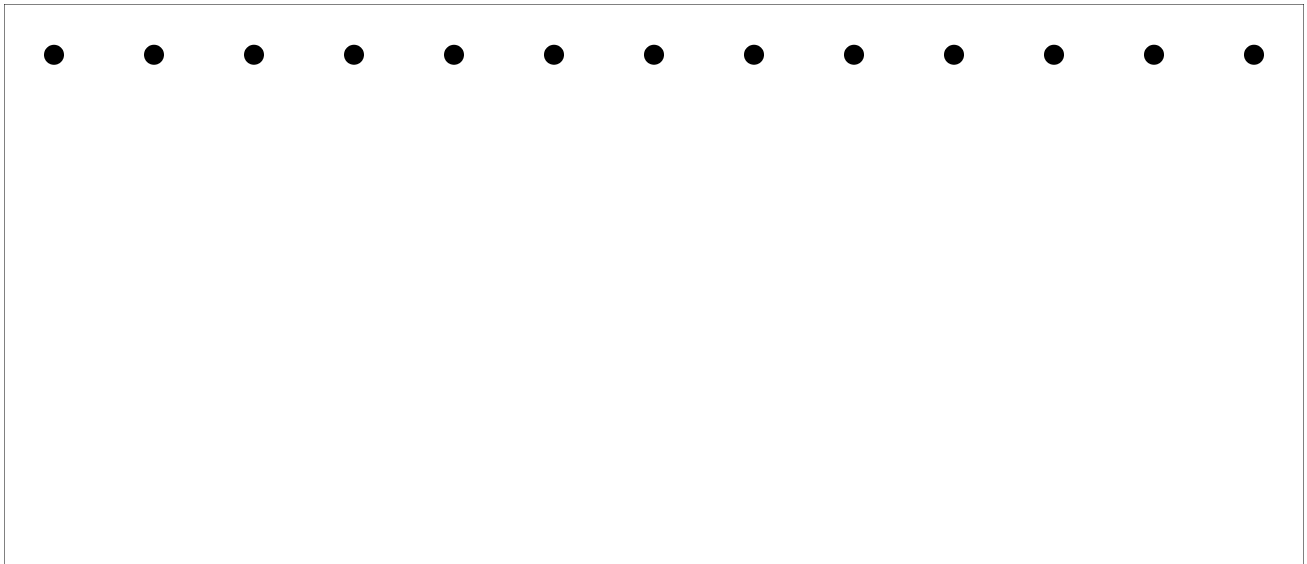
}

در این جا نیز نیروی ایجاد شده به خصوصیات ذره بستگی دارد و تنها دو ثابت ضریب گفته شده در مولد نگهداری می​شوند و به مانند جاذبه این مولد می​تواند یک نمونه اش برای چندین ذره به کار رود. این مدل مقاومت محیطی از آن damping که در خود particle​ها به کار می​رود بسیار پیچیده‌تر است و در موارد با دقت بالا و با حس واقعی‌تر می​تواند به کار رود.

## جمع​بندی و نتیجه​گیری

دیدیم که چگونه نیروهای مختلفی با جمع شدن می​توانند عمل نهایی شان را با هم به ذره وارد کنند. این نتیجه اصل دالامبر بوده که به ما این امکان را داد تا هر تعداد نیرو را بر هر تعداد ذره بدن دانستن این که نیرو چگونه به وجود می​آید اعمال کنیم. در فصل​های بعد تامین کنندگان نیروی دیگری و پیاده می​کنیم که در همین ساختاری که این جا پیاده کردیم می​گنجند.

در تصویر زیر یک نمایش از آنچه در این فصل توضیح داده شد را می​بینیم. در این مثال به تمام ذرات یک مولد نیروی جاذبه ثابت اعمال کرده‌ایم اما به هرکدام نیروی مقاومت هوای متفاوتی با ضریب مقاوت اول متفاوت و ضریب مقاوت دوم صفر داده‌ایم. همان طور که مشاهده می​کنیم این ذرات با آن که از نقطه ثابتی رها می​شوند اما به دلیل نیروی مقاومت هوای متفاوتی که بر آن​ها حاکم است، بعد از مدتی سرعتشان به طور متفاوت کاسته می​شود تا این که برخی با این که نیروی جاذبه شتاب می​خواهد در آن​ها ایجاد کند اما به دلیل نیروی مقاومت زیاد آن نیرو خنثی شده و به سرعت ثابت می​رسند.



شکل مثال مولدهای نیرو

# فصل هفتم: فنرها و خواص فنری

## مقدمه

یکی از کارامدترین نیروهایی که در موتور فیزیک می​توانیم پیاده کنیم، نیروی فنری است. فنر​ها برای نشان دادن اشیا نرم و قابل تغییر شکل دادن به کار می​روند. همچنین فنرها و ذرات می​توانند برای ایجاد طیف وسیعی از شکل​ها و پدیده​های فیزیکی به کار روند که از آن جمله می​توان طناب​ها، پرچمها و حتی موج دریا را ذکر کرد.

برای گنجاندن فنرها در این موتور فیزیک ابتدا تئوری فنرها مرور می​شود و سپس بررسی می​شود که چطور این تئوری می​تواند در موتور پیاده شود.

## قانون هوک

قانون هوک ریاضیات فنر​ها را در اختیار ما می​گذارد. این قانون بیان می​کند که نیروی ناشی از فنر بر جسم متصل به انتهای آن تنها ناشی از فاصله جسم از جایی است که فنر در حالت استراحت است. این فاصله می​تواند فاصله فشرده شده و یا گسترش یافته فنر باشد. فنری که دو برابر از دیگری کشیده شده باشد دو برابر بیشتر نیرو اعمال می​کند. معادله این قانون به این صورت است:



که در آن دلتای L نشان دهنده میزان اختلاف طول فنر از طول حالت عادی آن است و k ثابت فنر است که مقداری است که سختی فنر را تعیین می​کند. این نیرو در دو طرف فنر وارد می​شود. به عبارت دیگر اگر دو جسم در دو سوی فنر قرار گیرند به هر دو یک مقدار نیرو و با توجه به این معادله وارد می​شود. با توجه به این موضوع که در طول نرمال و استراحت فنر هیچ نیرویی وارد نمی​شود این معادله را می​توان به شکل زیر نوشت:



اما این معادله مربوط به زمانی است که در یک بعد فنر را بررسی می​کنیم، اگر بخواهیم آن را در دو بعد و با بردار​ها در نظر بگیریم این معادله به شکل زیر در خواهد آمد:



که در آن d از معادله زیر بدست می​آید:



که در آن​ها مختصات نقاط ابتدا و انتهای فنرها داده شده است. این معادله نشان می​دهد که نیرو باید در جهت انتهای دیگر فنر به جسم اعمال شود و با بزرگی ضریب فنر در مقدار خارج شدن آن از حالت نرمال. در واقع همین معادله را می​توان برای طرف دیگر فنر نیز به کار برد و نیروی وارده بر طرف دیگر را حساب کرد. چون این محاسبات یک بار تنها برای یک طرف صورت می​گیرد با دانستن طرف دیگر می​توان آن را برای آن هم به کار برد.

تنها فنر​ها اما نیستند که خاصیت فنری دارند و می​توانند از معادله هوک استفاده کنند. قانون هوک در تعداد زیادی از پدیده​های طبیعی عمل می​کند. در واقع هر چیزی که خاصیت ارتجاعی داشته باشد دارای محدودیت ارتجاعی هستند که در آن محدوده قانون هوک عمل می​کند. کاربردهایی از این دست بسیاز زیادند. طناب​های بانجی می​توانند با استفاده از این قانون پیاده شوند. نیروی ناشی از مایعات نیز با اتصال قسمت زیر آب رفته جسم با سطح آب توسط یک فنر فرضی می​تواند پیاده شود.

## مولد​های نیروی فنرگونه

در این جا به بررسی پیاده​سازی چند نوع مولد نیرو که با قانون هوک کار می​کنند می​پردازیم که تفاوت عمده آن​ها با هم نحوه محاسبه طول فعلی در قانون است.

این قسمت در واقع یک ویژگی بسیار مهم از سیستم​های فیزیکی را نشان می​دهد که آن یک هسته اصلی پردازش را به صورت عمومی​انجام می​دهد ولی توسط کلاس​ها و توابع کمکی احاطه شده که بسیار نزدیک به هم می​باشند و در این جا همان انواع تامین‌کنندگان نیروی فنرگونه می​باشند.

مولد نیرو فنری پایه طول فنر را به سادگی محاسبه کرده و با قانون هوک نیروی ناشی از آن را استخراج می​کند و به این صورت پیاده می​شود:

function ParticleSpring(other,springConstant,restLength){

ParticleForceGenerator.call(this);

this.otherParticle = other;

this.springConstant = springConstant;

this.restLength = restLength;

}

ParticleSpring.prototype = new ParticleForceGenerator();

ParticleSpring.prototype.constructor = ParticleSpring;

ParticleSpring.prototype.updateForce = function(particle,duration){

var force = new Point(particle.position.x,particle.position.y);

force.x -= this.otherParticle.position.x;

force.y -= this.otherParticle.position.y;

var magnitude = force.length;

magnitude = (magnitude - this.restLength);

magnitude \*= this.springConstant;

force = force.normalize();

force.x \*= -magnitude;

force.y \*= -magnitude;

console.log(force.y);

particle.addForce(force);

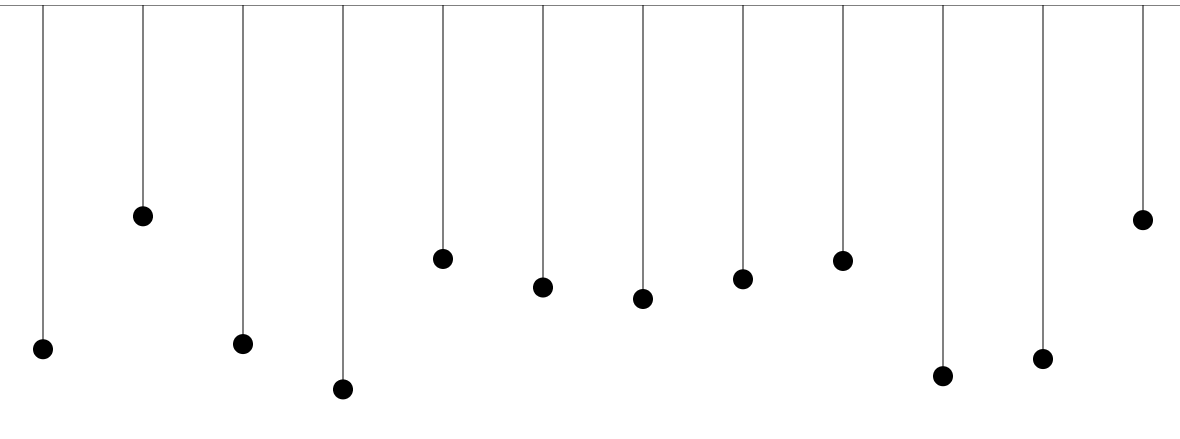
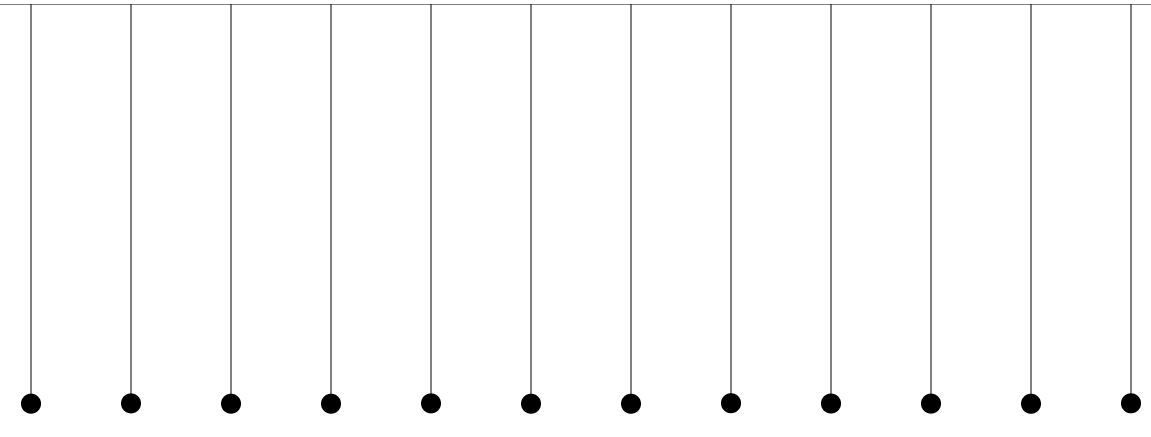
}

مولد با سه پارامتر ساخته می​شود. یکی ذره در انتهای دیگر است، دوم ثابت فنر و سوم طول استراحت فنر است.

این مولد چون برای هر فنر متفاوت است یک نسخه اش نمی​تواند برای هر ذره​​ای به کار رود و تنها برای یک ذره می​تواند به کار رود. همچنین این که برای دوسوی فنر لازم است که دو مولد جدا اما مثل هم ایجاد و به آن​ها اعمال شود.

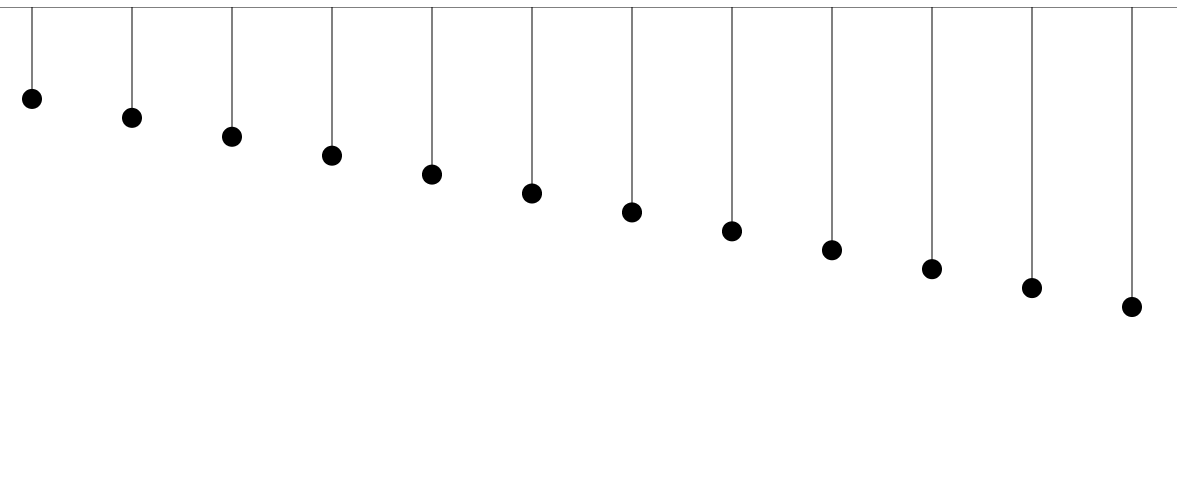
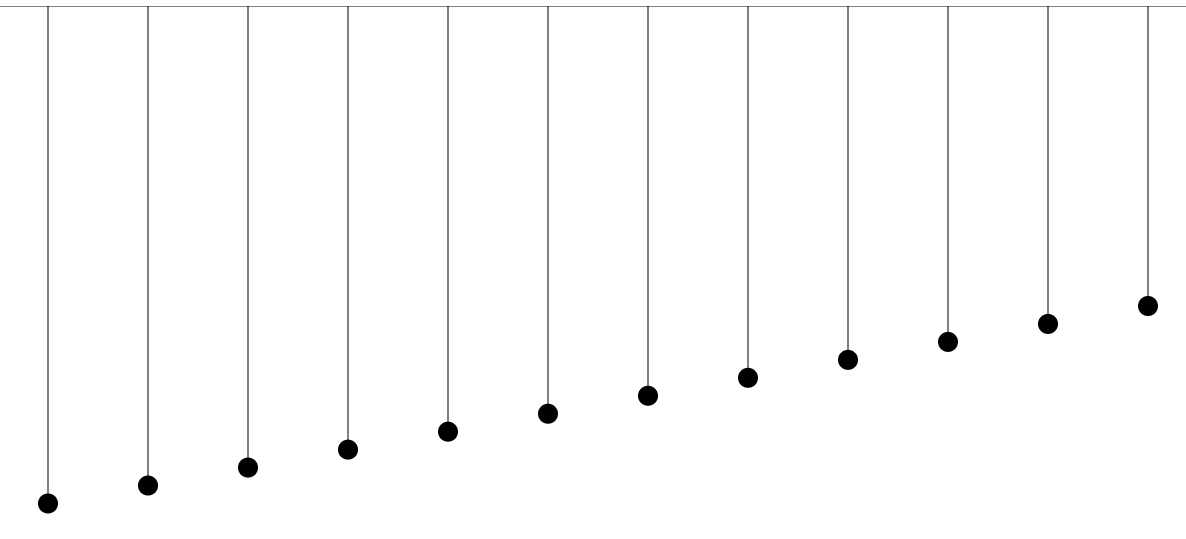
در بسیاری از مواقع نیازی به وجود دو جسم در دو طرف فنر وجود ندارد و نیاز هست که یک طرف فنر به جایی ثابت متصل باشد. در این صورت این نوع مولد نیرو که تا این جا درباره آن صحبت کرده‌ایم به کار نخواهد آمد و با تغییری جزئی یک نوع دیگر که این خاصیت را داشته باشد می​توانیم ایجاد کنیم. کد به گونه​​ای تغییر می​یابد تا به جای اتصال آن به یک شی دیگر به یک موقعیت برداری متصل شود و طول​ها از آن محاسبه شوند.

در شکل زیر یک مثال استفاده از مولد نیروی فنر را مشاهده می کنید. در این مثال به چندین فنر که هر کدام یک ثابت قانون هوک متفاوت دارند و یک انتهای آن​ها به بالای صفحه متصل شده است یک جسم​هایی با جرم​های یکسان متصل شده‌اند. همگی فنر​ها دارای طول استراحت یکسان و در لحظه ابتدایی حرکت از یک طول باز هم یکسان رها می شوند. اما همان طور که در تصویر دوم مشخص است داشتن ثابت​های فنری متفاوت برای فنرها با آنکه تمام شرایط دیگر یکسان است باعث می شود که رفتار متفاوت و متناسب با ثابت خود نشان دهند:



شکل فنرها

در یک مثال دیگر که تصویر آن در پایین آمده است می توان مشاهده کرد که چگونه کشش متفاوت یک نوع فنر در چندین شکل مختلف به رفتار متناسب و در نسبت با مقدار انحراف از طول استراحت فنر می انجامد. همچنین یک نکته پنهان در مثال زیر نگهداری انرژی جنبشی در فنر است که با توجه به تغییر مقدار سرعت با توجه به موقعیت جسم مشاهده می شود:



شکل مثال دوم فنرها

یک طناب بانجی ارتجاعی می​تواند نیروی کششی ایجاد کند، می​توان آن را فشرده کرد اما نیروی پرت کننده​​ای ایجاد نخواهد کرد اما مانند هر فنر دیگری وقتی کشیده می​شود نیروی کششی ایجاد می​کند. این خاصیت برای نگهداری دو جسم در فاصله​​ای از هم مفید است. دو جسم اگر از هم بیش از حد دور شوند به صرف هم کشیده می​شوند اما هر چقدر که بخواهند می​توانند به هم نزدیک شوند بدون این که از هم جدا شوند. این مولد به این شکل پیاده می​شود:

function ParticleBungee(other,springConstant,restLength){

ParticleForceGenerator.call(this);

this.otherParticle = other;

this.springConstant = springConstant;

this.restLength = restLength;

}

ParticleBungee.prototype = new ParticleForceGenerator();

ParticleBungee.prototype.constructor = ParticleSpring;

ParticleBungee.prototype.updateForce = function(particle,duration){

var force = new Point(particle.position.x,particle.position.y);

force.x -= this.otherParticle.position.x;

force.y -= this.otherParticle.position.y;

var magnitude = force.length;

if(magnitude<=this.restLength){

return;

}

magnitude = Math.abs(magnitude - this.restLength);

magnitude \*= this.springConstant;

force = force.normalize();

force.x \*= -magnitude;

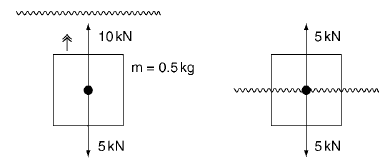
force.y \*= -magnitude;

particle.addForce(force);

}

همان طور که برای فنر​ها نیز یک نسخه تک جسمی​ایجاد کردیم برای bungee هم می​توانیم این مولد نیرو را بدون نیاز به جسم دیگر ایحاد کرده و طرف دیگر را به یک موقعیت در صفحه متصل کنیم که ثابت می​باشد.

نیروی شناوری نیرویی است که باعث شناور ماندن جسم در مایع می​شود. ارشمیدس برای اولین بار بیان کرد نیروی شناوری با مقدار مایعی که جسم با داخل شدن به مایع جابجا می​کند نسبت مستقسم دارد. در شکل زیر مشاهده می​کنیم که یک جسم به جرم 0.5 کیلوگرم در آب فرو رفته است.​​ای خالط چگالی 1000 کیلوگرم بر متر مکعب را دارد، یعنی یک متر مکعب از آب جرم یک تن را دارد. جسم شکل زیر 0.001 متر مکعب حجم دارد، بنابراین همین مقدار آب را جابجا می​کند. بنابراین این مقدار آب یک کیلوگرم جرم خواهد داشت.



شکل نیروهای شناوری

باید توجه داشت که جرم به معنی وزن نیست و وزن نیرویی است که به واسطه جاذبه بر جرم ایجاد شده است و با توجه به تغییر جاذبه می​تواند تغییر کند هر چند که جرم همیشه ثابت است و از معادله زیر بدست می​آید:

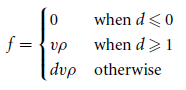


بر روی زمین مقدار g 10 فرض می​شود. بنابراین جسم مورد نظر در شکل سمت چپ که کاملا در آب فرور رفته است نیروی 10 kN را از طرف مایع به خود می​بیند. اما جسم سمت راست که به طور نسبی در آب فرو رفته است تنها 5 kN نیرو را از طرف آب دریافت می​کند. وزن جسم در هر دو شکل یکسان است و چون در سمت چپ کمتر از نیروی شناوری است به سمت بالا حرکت می​کند اما وقتی به وضعیت شکل سمت راست می​رسد نیروها با هم برابر شده و جسم حالت ایستایی پیدا می​کند و شناور می​ماند.

محاسبه مقدار دقیق نیروی شناوری بستگی به دانش در رابطه با شکل جسم دارد، چرا که این شکل جسم است که تعیین می​کند چقدر از مایع جابجا شده است که در محاسبه نیروی ناشی از آن نقش داشته باشد. اما در واقع آن چنان که به واقع نیاز هست، این مقدار جزئیات عملا در موتور فیزیکی لازم نیست، با محاسبه تقریبی شکل جسم به جسم​های ساده هندسی و به خصوص مستطیل​های دوری هر شکلی می​توان به حس نیروی شناوری نسبتا دقیقی رسید.

در نتیجه می​توانیم از یک تقریب فنرگونه برای محاسبه این نیرو استفاده کنیم. وقتی شیئ در نزدیکی سطح آب است از یک نیروی مانند فنر استفاده می​کنیم تا نیروی شناوری را بتوانیم ایجاد کنیم. این نیرو با میزان عمق فرو رفتگی جسم در مایع نسبت مستقیم دارد، همان طور که در فنر نیروی حاصله با مقدار کشیدگی و فشردگی از اندازه نرمال نسبت مستقیم دارد. همان طور که در شکل هم دیدیم این روش برای شیئ مستطیلی که کاملا در آب فرو نرفته است دقیق است. برای هر نوع شیئ دیگر اما یک تقریب خوبی است که هر چند دقیق نیست.

وقتی جسم کاملا در آب فرو رفته باشد، رفتار متعاقب با آن کمی​متفاوت خواهد بود. در این حالت عمیق‌تر رفتن آن به داخل آب باعث نخواهد شد که آب بیشتری جابجا شود، بنابراین از آن جایی که آب چگالی یکسانی خواهد داشت این کار نیروی وارده را افزایش نخواهد داد. اما یک نکته این است که سیستم ذره​​ای که ما استفاده می​کنیم فاقد اندازه است، بنابراین نمی​توانیم با همین اطلاعات مشخص کنیم که چه زمانی جسم کاملا در آب فرو رفته است، اما می​توانیم برای مولد نیروی شناوری آب یک اندازه مشخص در رابطه با ذره مشخص در نظر بگیریم و آن را به عنوان حداکثر عمق به کار ببندیم. بنابراین مولد برای فرو رفتن ذره بیش از آن مقدار نیروی بیشتری تولید نخواهد کرد. از طرف دیگر درست برعکس این موضوع نیز می​باشد چرا که وقتی جسم به بالا حرکت می​کند به جایی می​رسد که فقط قسمتی از آن در آب است و قسمت دیگر بیرون از آب است، اما وقتی این حرمت را ادامه دهد به نقطه​​ای خواهد رسید که تمام آن از آب بیرون خواهد رفت، در این حالت اهمیت دارد که هیچ نیرویی از طرف این مولد نیرو ایجاد و به ذره وارد نشود و هرچقدر هم این بالا رفتن و فاصله گرفتن از آب بیشتر شود این موضوع باید صادق بماند. بنابراین می​توان معادله محاسبه نیرو را به این شکل نشان داد:



که در آن p چگالی مایع، v حجم شیئ و d مقدار عمقی است که شیئ در مایع فرورفته است که به صورت نسبتی از کل عمق فرورفتگی کامل جسم بیان می​شود. وقتی کاملا جسم در مایع فرو رفته باشد، d بزرگتر یک و وقتی کاملا بیرون از مایع باشد d کوچکتر از صفر خواهد بود که از معادله زیر بدست می​آید:



که در آن s حدالثر عمق غرق شدگی جسم و y مختصات شیئ و مایع را مشخص می​کند.

تمام این مباحث در کد زیر پیاده شده است:

function ParticleBuoyancy (maxDepth,volume,waterHeight,liquidDensity){

ParticleForceGenerator.call(this);

this.maxDepth = maxDepth;

this.volume = volume;

this.waterHeight = waterHeight;

this.liquidDensity = liquidDensity;

}

ParticleBuoyancy.prototype = new ParticleForceGenerator();

ParticleBuoyancy.prototype.constructor = ParticleBuoyancy;

ParticleBuoyancy.prototype.updateForce = function(particle,duration){

var depth = particle.position.y;

if (depth >= this.waterHeight + this.maxDepth){

return;

}

var force = new Point(0,0);

if (depth <= this.waterHeight - this.maxDepth)

{

force.y = this.liquidDensity \* this.volume;

particle.addForce(force);

return;

}

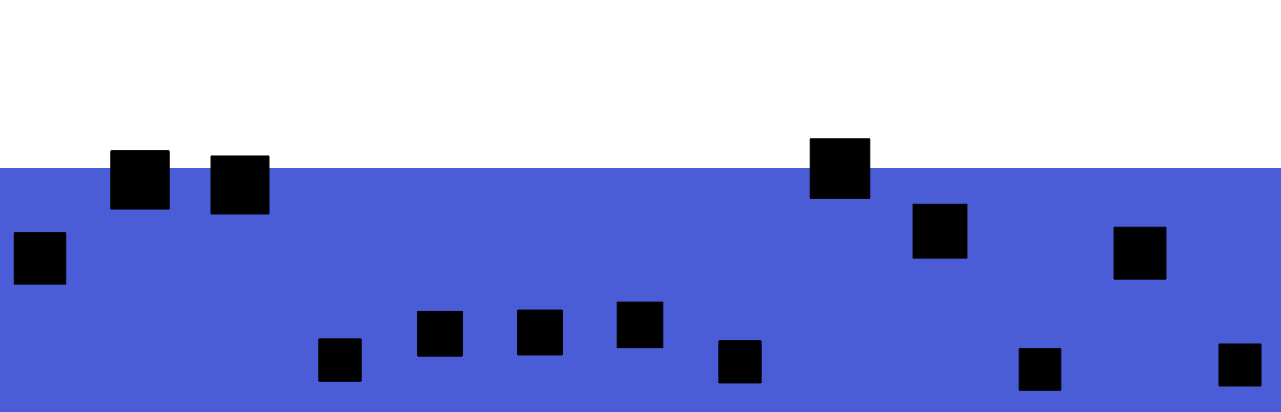
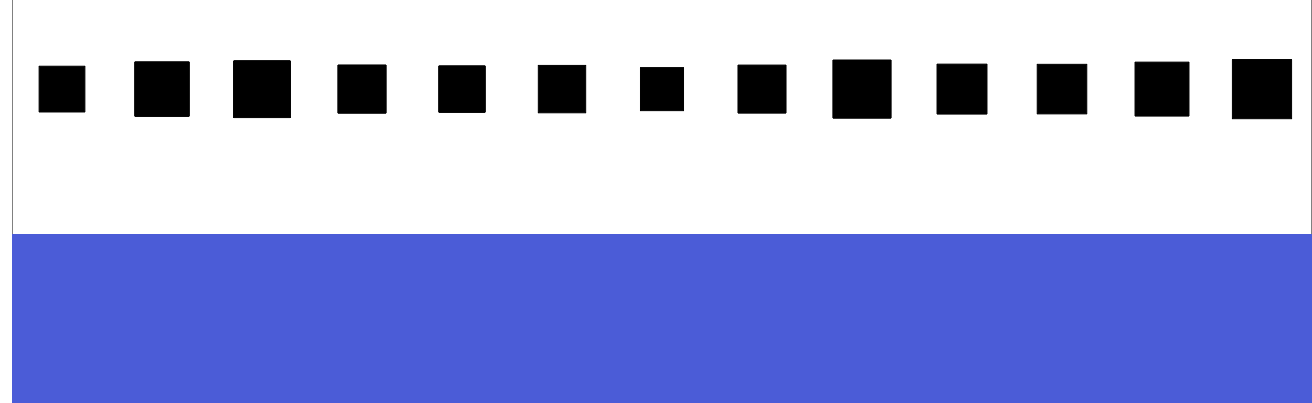
force.y = this.liquidDensity \* this.volume \*(depth - this.maxDepth - this.waterHeight) / 2 \* this.maxDepth;

particle.addForce(force);

}

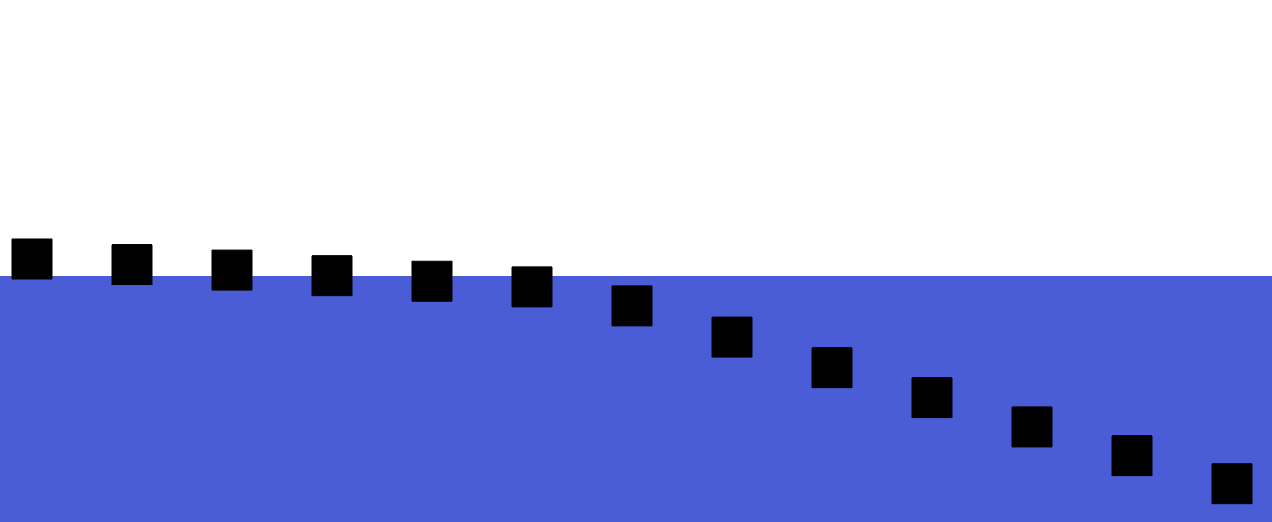
مولد بالا چهار پارامتر را دریافت می​کند که عبارتند از بیشترین عمق فرورفتگی، حجم جسم، عمق از سطح آب و چگالی مایعی که جسم در آن غوطه ور است. با توجه به این پارامتر​ها مشخص است که این مولد تنها به یک ذره می​تواند اعمال شود و برای هر ذره باید مولد خاص خود را ایجاد نماییم.

در شکل​های زیر یک مثال از مولد نیروی شناور آورده شده است که در آن تعدادی جسم با رفتار ذره از یک ارتفاع یکسان در یک مایع انداخته می شوند. تمام پارامترهای مربوط به نیروی مولد از جمله چگالی مایع، ارتفاع مایع و عیره یسان می باشند. آن چه برای هر جسم متفاوت است و به صورت رندم ایجاد می شود حجم هر جسم است که با توه با آن شکل آن نیز بزرگتر یا کوچکتر و در نتیجه ارتفاع فورفتگی آن هم متفاوت است، آن چنان که مشخص است، چگونه این پارامترها در رفتار مایع بر جسم شناور تاثیر با نسبت گفته شده می گذارد و به طور خلاصه نیروی:



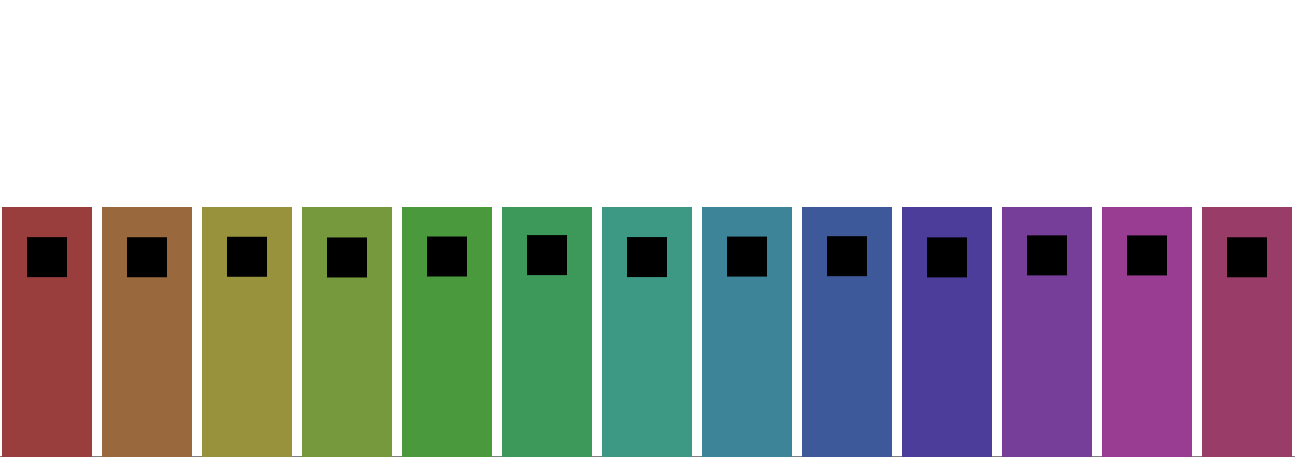
شکل مثال اول نیروهای شناوری

در یک مثال دیگر می توان مشاهده کرد که افزایش یک جسم ثابت با تمام متغیرهای دیگر ثابت چگونه در فرورفتگی جسم تاثیر گذار است. آن چنان که در تصویر زیر قابل مشاهده است، افزایش جرم جسم موجب بالا رفتن چگالی آن و در نتیجه بیشتر فرو رفتن جسم در آب می شود چنان که از یک حد به بعد این موضوع باعث فرو رفتن و غرق شدن کامل آن می شود.



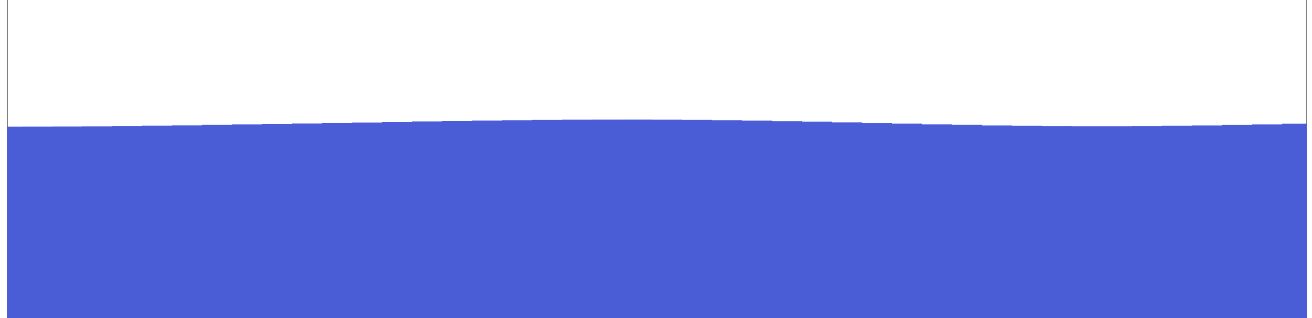
شکل مثال سوم نیروهای شناوری

در مثال بعد تاثیر تغییر مایع و در نتیجه چگالی آن را بر جسم​های کاملا یکسانی که از ارتفاع یکسانی در داخل مایع رها می شوند را مشاهده می کنیم که چگالی آن​ها چطور بر رفتار یک جسم ثابت و نیروی وارد بر آن تاثیر می گذارد:



شکل مثال دوم نیروهای شناوری

اما در یک مثال دیگر یک نوع استفاده دیگر از اجسام شناور و مایع کردیم و آن ایجاد موج​هایی بر روی آب با استفاده از اجسام شناور به هم چسبیده است که این پدیده را به خوبی شبیه‌سازی می کنند، تصاویر این شبیه‌سازی در این جا آورده شده است:



شکل مثال چهارم نیروهای شناوری

## جمع​بندی و نتیجه​گیری

پس به طور خلاصه مشاهده کردیم که چگونه تعداد زیادی از پدیده​های فیزیکی را می​توان با خاصیت قانون هوک شبیه‌سازی کرد. بعضی از آن​ها مانند نیروی شناوری کاملا مشابه نیروی فنری عمل می​کنند. در نتیجه با توجه به این موضوع یک دسته از تامین کنندگان انرژی را پیاده​سازی کردیم. لازم به ذکر است که می​توان حتی برخورد​ها و نفوذ بسیاری دیگر از پدیده​هایی که در فصل​های بعد بررسی می​کنیم را نیز با این قانون پیاده کرد اما از آنجا که قانون هوک و شبیه‌سازی آن در موارد جزئی ضعیف عمل می​کند و در برخی موارد دقت لازم را ندارد این موارد را با ابزار​های دیگری پیاده می​کنیم که در دو فصل بعد به آن​ها می​پردازیم.

# فصل هشتم: محدودیت​های قوی

## مقدمه

با وجود این که با قانون فنر می​توان وضعیت​های مختلفی را شبیه‌سازی کرد باشد، در عین حال می​تواند رفتار نامناسبی از خود نشان دهد. برای نمونه، شبیه‌سازی ضریب فنر، به هنگام بستن محکم اجسام به هم، تقریبا غیر ممکن است. در شرایطی که اجسام توسط میله​های محکم به هم متصل می​شوند و یا با استفاده از سطوح سخت از هم جدا نگه داشته شده اند، فنرگزینه‌ی بادوامی​نیست.

در ابتدا نگاهی می​اندازیم به معمول​ترین محدودیت قوی، یعنی برخورد​ها و تماس بین اجسام. ریاضیات مربوط به این محدودیت، می​تواند برای انواع دیگر محدودیت​ها، همچون میله​ها یا کابل​های غیر قابل بسط که برای اتصال اجسام استفاده می​شوند، به کار رود.

برای مدیریت محدودیت​های قوی در موتور فیزیک پیاده​سازی شده، نیاز به کنار گذاشتن دنیای مولدهای نیرو داریم. در تمام موتور​های فیزیک پیاده​سازی شده در این پروژه، بین مولدهای نیرو و محدودیت​های قوی تفاوت قائل شده است.

## تحلیل ساده برخورد

برای مدیریت محدودیت​های قوی، سیستمی​به منظور تحلیل برخورد به موتور اضافه شده است. برخورد به هر وضعیتی که در آن دو جسم با هم درتماس باشند، گفته می​شود. در زبان عامیانه برخورد به فرایندی خشن تلقی می​شود که دو جسم در آن با سرعت نزدیک شونده قابل ملاحظه​​ای با هم مواجه می​شوند. اما در نگاهی که در این موتور فیزیک به برخورد شده، علاوه بر این حالت، تماس دو جسم بدون سرعت نزدیک شونده نیز برخورد تلقی می​شود. فرایندی مشابه برای تحلیل برخوردهای با سرعت زیاد و تماس ایستا به کار برده خواهد شد. به همین دلیل، در این جا دو لغت تماس و برخورد مفهوم یکسانی خواهند داشت.

زمانی که دو جسم با هم برخورد می​کنند، حرکت بعد از برخورد این دو جسم می​تواند با توجه به حرکت قبل از زمان برخورد محاسبه شود. این عمل، تحلیل برخورد نام دارد. برخورد به گونه​​ای تحلیل می​شود که اطمینان حاصل شود که هر دو جسم بعد از برخورد، دارای حرکت صحیحی هستند. فرایند برخورد درمدت زمان بسیار کوتاهی رخ می​دهد (به همین دلیل، ما قادر به دیدن فرایند برخورد بسیاری از اجسام نیستیم).

قوانین حاکم بر حرکت دو جسم در حال برخورد، بستگی به سرعت نزدیک شدن آن دو جسم دارد. سرعت نزدیک شدن، سرعت کلی است که دوجسم روی هم رفته، با آن سرعت حرکت می​کنند. دقت شود که در این جا از لفظ سرعت نزدیک شدن (velocity closing) استفاده شد نه سرعت (speed)، چرا که سرعت کمیتی عددی است نه برداری، در نتیجه سرعت فاقد جهت است و تنها می​تواند مقادیر مثبت یا صفر داشته باشد، این در حالی است که سرعت نزدیک شدن می​تواند جهت داشته باشد. برای نمونه، دو جسمی​که در جهت دور شدن از هم حرکت می​کنند، سرعت نزدیک شدنی کمتر از صفر دارند. سرعت نزدیک شدن دو جسم با یافتن مولفه سرعت در جهت جسمی​به جسم دیگر محاسبه می​شود:



در رابطه‌ی بالا، cv نشانگر سرعت نزدیک شدن (کمیتی مقداری) است، ap و bp موقعیت دو جسم a وb را نشان می​دهند. نقطه(.) نشان دهنده ضرب عددی، و p بردار طول واحد در جهت خود p است.

این معادله می​تواند به صورت زیر ساده شود. عموما علامت این مقدار عوض می​شود. در نتیجه معادله، به جای سرعت نزدیک شدن، سرعت دور شدن را مشخص می​کند. سرعت نزدیک شدن برابر سرعت حرکت یک جسم نسبت به جسم دیگر، در جهت بین دو جسم است.



در معادله جدید، دو جسمی​که در حال نزدیک شدن هستند دارای سرعت نسبی منفی و دو جسمی​که در حال دور شدن هستند داری سرعت نسبی مثبت هستند. در واقع از نظر ریاضی، این کار معادل عوض کردن علامت در معادله قبل است.

زمانی که دو جسم با هم برخورد می​کنند، به هم فشرده می​شوند و تغییر شکل فنر مانند سطوح دوجسم، نیروهایی ایجاد می​کند که باعث دور شدن دو جسم خواهد شد. تمام این اتفاقات در مدت زمان کوتاهی رخ می​دهد (این اتفاقات آنقدر سریع رخ می​دهد که امکان شبیه‌سازی فریم به فریم آن وجود ندارد). سرانجام دو جسم سرعت نزدیک شونده خود را از دست خواهند داد. با وجود این که این رفتار به مانند رفتار فنر است، اما در واقعیت اتفاقات دیگری نیز می​افتد. در زمان فشرده شدن، هر اتفاقی می​تواند رخ دهد وخصوصیات مواد درگیر، می​تواند موجب بروز عکس العمل​های پیچیده شود. در واقعیت این رفتار با رفتار فنر مطابقت ندارد.

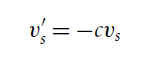
مدل فنر فرض می​کند که مقدار تکانه در طول مدت زمان برخورد، نگهداری می​شود:



در این معادله، am نشان دهنده جرم جسم a، ap**.** سرعت جسم a قبل از برخورد، و a'p**.** سرعت جسم a بعد از برخورد است.

خوشبختانه، بیشتر برخوردها رفتاری مشابه فنر ایده آل دارند. می​توان با فرض نگه داشته شدن مقدار جنبش آنی، رفتاری کاملا قابل باور ایجاد کرد و از معادله قبل برای مدل کردن برخورد استفاده کرد.

معادله قبل، سرعت کل قبل و بعد از برخورد را مشخص می​کند. اما از این رابطه نمی​توان سرعت هر یک از اجسام را جداگانه محاسبه کرد. سرعت​های جداگانه از طریق سرعت نزدیک شدن به هم مرتبط هستند. طبق معادله:



که در آن، s'v نمایانگر سرعت دور شدن بعد از برخورد، sv سرعت دور شدن قبل از برخورد است و c ثابتی است به نام ضریب ارتجاع.

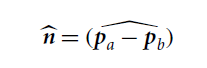
ضریب ارتجاع، سرعتی را که در آن، اجسام بعد از برخورد شروع به دور شدن می​کنند، تحت کنترل دارد. این ضریب بستگی به مواد درگیر در برخورد دارد. دو ماده متفاوت، ضرایب متفاوتی دارند. برخی از اجسام مانند توپ تنیس، بعد از برخورد با راکت، به دور پرتاب می​شوند. اما اجسام دیگر بعد از برخورد به هم می​چسبند. مانندگلوله برفی پس از برخورد با صورت.

اگر این ضریب برابر یک باشد، اجسام با همان سرعتی که با آن به هم نزدیک شده بودند، به دور از هم پرتاب می​شوند اما اگر این ضریب برابر صفر باشد، دو جسم یکی می​شوند و با هم حرکت می​کنند (یعنی سرعت دور شدشان برابر صفر است). حتی با صرف نظر از ضریب ارتجاع نیز، رابطه قبل‌تر همچنان برقرار است، یعنی مقدار کل تکانه همان خواهد بود. با استفاده از هر دو معادله، می​توان مقادیر ap**.** و bp**.** را محاسبه کرد.

تا این جا در مورد مولفه​های برخورد بین دو جسم صحبت شد. اما گاهی اوقات، قصد پیاده​سازی برخورد بین یک جسم و چیزی که به طور فیزیکی قابل شبیه‌سازی نیست را داریم. این جسم ممکن است زمین، دیوارهای یک سطح و یا هر جسم فاقد قدرت حرکت باشد. می​توان این اجسام را به صورت اجسام با جرم بی نهایت نمایش داد ولی این کار سودی نخواهد داشت. مطابق تعریف این اجسام هرگز حرکت نمی​کنند.

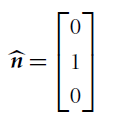
اگر بین یک جسم و قسمتی از یک صحنه فاقد حرکت برخوردی صورت گیرد، نمی​توان سرعت دور شدن را بر اساس بردار بین مکان دو جسم محاسبه کرد، زیرا تنها یک جسم وجود دارد. به عبارت دیگر، نمی​توان از لفظ (bp-ap)- در رابطه تکانه​ها استفاده کرد. باید برای آن جایگزینی یافت. عبارت (bp-ap)- بیانگر جهت سرعت دور شدن است. سرعت دور شدن با استفاده از ضرب نقطه​​ای سرعت​های نسبی دو جسم واین عبارت محاسبه می​شود. اگر دو جسم وجود نداشته باشد، باید جهت به طور مشخص و صریح گفته شود. این جهت نشان دهنده جهتی است که دوجسم با هم برخورد می​کنند و عموما به آن نرمال برخورد یا نرمال تماس گفته می​شود. از آن جایی که این نرمال نشان‌دهنده‌ی جهت است، این بردار دارای بزرگی یک است.

در شرایطی که دو ذره داریم که با هم برخورد می​کنند، نرمال تماس با استفاده از رابطه زیر به دست می​آید.



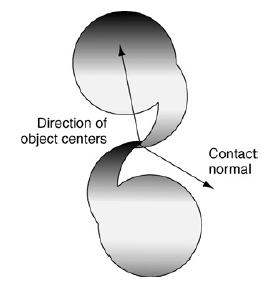
به طور قرارداردی، همواره نرمال تماس را از زاویه دید جسم a در نظر گرفته می​شود. در این مثال، از دید جسم a، تماس از جانب جسم b صورت می​گیرد. در نتیجه از عبارت bp-ap استفاده می​شود. اگرجهت برخورد از نگاه جسم b، در نظر گرفته شود، می​توان به سادگی عبارت را در ١- ضرب کرد.در عمل، از این روش به طور صریح استفاده نمی​شود بلکه، عمل معکوس کردن را داخل کد مورد استفاده برای محاسبه سرعت دور شدن جسم b می​آوریم. بعدا در بیان پیاده​سازی این نکته قابل مشاهده است که علامتی منفی در محاسبات مربوط به جسم b آورده می​شود.

زمانی که ذره​​ای با زمین بر خورد می​کند، ما فقط جسم a را داریم و جسم b وجود ندارد. در این شرایط از دید جسم a، نرمال تماس



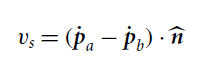
خواهد بود با این فرض که سطح در زمان برخورد هموار است.

اگر از ذرات بگذریم و بر روی اجسام صلب تمرکز کنیم، وجود نرمال تماس صریح، بسیار سخت خواهد بود. شکل زیر نمونه​​ای از شرایط ممکن را نشان می​دهد. در این جا، دو جسم در حال برخورد، به موجب نوع شکلشان، نرمال تماسشان دقیقا در جهت عکس آن چیزی است که تصور می​شود.. دو جسم به شکل قوس روی هم حرکت می​کنند و نرمال تماس به گونه​​ای رفتار می​کند که گویی مانع جدا شدن اجسام است تا تماس پیدا کردن. بعدا نمونه​های دیگری آورده می​شود که می​تواند برای نمایش میله و یا اتصالات سخت دیگر به کار رود.



شکل خطای نرمال

با داشتن نرمال تماس صحیح، معادله سرعت نزدیکی به صورت زیر در می​آید:



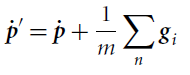
تنها تغییری که برای تحلیل برخورد لازم است، تغییر سرعت است. تا این جا در این موتور فیزیک، تنها روشی که برای تغییر سرعت به کار برده شده است، تغییر شتاب بوده است. اگر تغییر شتاب برای مدت زمانی طولانی اعمال شود، سرعت به شدت تغییر خواهد کرد. اما در مورد مطرح شده در این جا، تغییرات لحظه​​ای هستند. به همین دلیل سرعت نیز به صورت آنی مقادیرجدید به خود می​گیرد. لازم به ذکر است که اعمال نیرو سبب تغییر شتاب یک جسم می​شود. اگر ما نیرو را فورا تغییر دهیم شتاب نیز فورا تغییر خواهد کرد. در رابطه با تغییر سرعت نیز می​توان چیزی مشابه در نظر گرفت. اما به جای نیرو به آن ضربه گفته می​شود. ضربه یعنی تغییری آنی و فوری در سرعت. همان گونه که در مورد نیرو رابطه‌ی برقرار است:



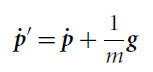
در رابطه با ضربه نیز رابطه زیرصدق می​کند:



یک تفاوت اساسی بین نیرو و ضربه وجود دارد. جسم فاقد شتاب است مگر این که نیرویی به آن وارد شود یعنی می​توان شتاب کل را با استفاده از قانون دالامبر از طریق ترکیب تمام نیروها محاسبه کرد. اما اگر به جسم نیرو یا ضربه​​ای نیز وارد نشود باز هم سرعت خواهد داشت. ضربه سرعت را تغییر می​دهد. می​توان ضربه را نیز با استفاده از قانون دالامبر ترکیب کرد. اما نتیجه برابر مجموع تغییرسرعت خواهد بود نه کل سرعت:



١g تا ng مجموعه‌ی تمام ضربه​های اعمال شده بر جسم است. در واقعیت، ضربه​ها را به گونه​​ای که نیرو​ها را جمع کردیم، جمع نمی​کنیم. ضربه​ها در طول فرایند برخورد، هر زمان که رخ دهند، اعمال می​شوند. هر یک از ضربه​ها در زمانی خاص با استفاده از معادله زیر، اعمال می​شوند.



نتیجه‌ی تحلیل برخورد، ضربه​​ای است که بر هر جسم وارد می​شود. ضربه فورا بر جسم اعمال می​شود و سرعت به طور آنی تغییر می​کند.

به منظور مدیریت برخوردها از کد زیر استفاده می​کنیم. وظیفه lvertactResoCon گرفتن مجموعه​​ای ازبرخوردها و اعمال ضربه​های مناسب به اجسام درگیر در برخورد است. هر برخورد در یک ساختمان داده Contact آماده شده است که به شکل زیر است:

function ParticleContact()

{

this.particle = new Array();

this.restitution = null;

this.contactNormal = new Point();

}

این ساختمان داده اشاره گری به هر یک از اجسام درگیر برخورد نگه می​دارد: یک بردار که حاوی نرمال تماس از دید جسم اول است و عضو داده​​ای برای نگه داری ضریب ارتجاع تماس. اگر برخورد بین یک جسم و یک صحنه رخ دهد (تنها یک جسم وجود داشته باشد)، در این صورت اشاره گر جسم دوم برابرNULL خواهد بود.

برای تحلیل یک تماس، معادلات برخوردی که قبلا ملاحظه شد را پیاده​سازی می​کنیم.

ParticleContact.prototype.resolve = function(duration){

this.resolveVelocity(duration);

}

ParticleContact.prototype.calculateSeparatingVelocity = function(){

var relativeVelocity = this.particle[0].velocity;

if (this.particle[1]){

relativeVelocity.x -= this.particle[1].velocity.x;

relativeVelocity.y -= this.particle[1].velocity.y;

}

return relativeVelocity.scalarProduct(this.contactNormal);

}

ParticleContact.prototype.resolveVelocity = function(duration){

var separatingVelocity = this.calculateSeparatingVelocity();

if (separatingVelocity > 0)

{

return;

}

var newSepVelocity = -separatingVelocity \* this.restitution;

var deltaVelocity = newSepVelocity - separatingVelocity;

var totalInverseMass = this.particle[0].inverseMass;

if (this.particle[1]){

totalInverseMass += this.particle[1].inverseMass;

}

if (totalInverseMass <= 0) return;

var impulse = deltaVelocity / totalInverseMass;

var impulsePerIMass = new Point(this.contactNormal.x \* impulse,this.contactNormal.y \* impulse);

this.particle[0].velocity.x = this.particle[0].velocity.x + impulsePerIMass.x \* this.particle[0].inverseMass;

this.particle[0].velocity.y = this.particle[0].velocity.y + impulsePerIMass.y \* this.particle[0].inverseMass;

if (this.particle[1])

{

this.particle[1].velocity.x = this.particle[1].velocity.x + impulsePerIMass.x \* -this.particle[1].inverseMass;

this.particle[1].velocity.y = this.particle[1].velocity.y + impulsePerIMass.y \* -this.particle[1].inverseMass;

}

}

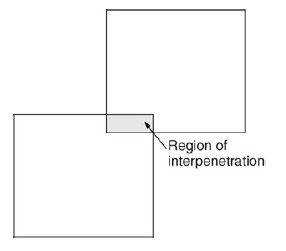
این کد، سرعت هر یک از اجسام درگیر را به منظور انعکاس برخورد، تغییر می​دهد.

نقاط برخورد معمولا توسط یک یابنده برخورد مشخص می​شود. یابنده برخورد تکه کدی است که مسئول یافتن دو جسم در حال برخورد یا جسم در حال برخورد با یک صحنه‌ی فاقد قابلیت حرکت است. در این موتور فیزیک، نتیجه‌ی الگوریتم تشخیص برخورد، مجموعه​​ای از ساختمان داده​های Contact است که با اطلاعات مناسب پر شده است. تشخیص برخورد نیاز به دانستن هندسه اجسام، یعنی شکل و اندازه اجسام دارد. تا این جا در این موتور فیزیک، اجسام به صورت ذره فرض شده است که اصلا هندسه​​ای برای آن در نظر گرفته نشده است.

سیستم شبیه‌ساز فیزیکی (قسمتی از موتورکه مسئولیت مدیریت قوانین حرکت، تحلیل برخورد و نیروها را بر عهده دارد)، نیازی به آگاهی از جزئیات شکل اجسام درگیر برخورد ندارد. سیستم تشخیص برخورد مسئولیت محاسبه هرگونه خصوصیت هندسی را بر عهده دارد. برای نمونه تشخیص این که دو جسم درچه زمان و مکانی با هم تماس پیدا می​کنند و مشخص کردن نرمال تماس بین دو جسم.

برخی از الگوریتم​های تشخیص برخورد، شیوه‌ی حرکت اجسام را زیر نظر می​گیرند و تلاش می​کنند برخورد​های احتمالی آینده را پیش بینی کنند. به این صورت که به مجموعه اجسام نگاه می​اندازند و بررسی می​کنند که آیا هیچ دو جسمی​از هم خواهند گذشت یا خیر.

دو جسم با هم تداخل دارند اگر به صورت جزئی به مانند شکل زیر در هم ادغام شده باشند:



شکل تداخل

در زمان پردازش برخوردی که دو جسم درهم فرو می​روند، تنها تغییر سرعت کافی نیست. اگر اجسام در این حالت با ضریب ارتجاع نسبتا کوچکی با هم برخورد کنند، سرعت دورشدنشان تقریبا برابر صفر خواهد بود.در این حالت دو جسم هیچ گاه از هم جدا نمی​شوند و فرد بازیکن مشاهده می​کند که اجسام در هم گیر کرده اند و جدا‌سازی آن​ها غیر ممکن خواهد بود.

به عنوان جزیی از تحلیل برخورد، نیاز به تحلیل تداخل داریم. وقتی دو جسم در هم نفوذ می​کنند، تنها به اندازه​​ای دو جسم را از هم دور می​کنیم که از هم جدا شوند. این وظیفه یابنده برخورد است که به عنوان بخشی از ساختمان داده Contact مشخص کند دو جسم چقدر در هم نفوذ کرده‌اند. محاسبه مقدار نفوذ دو جسم بستگی به هندسه اجسام درگیر در برخورد دارد. همان طور که دیده شد این کار وظیفه سیستم یابنده برخورد است نه شبیه‌ساز فیزیک.

برای نگه داشتن این اطلاعات، یک عضو داده​​ای به ساختمان داده Contact به صورت زیر اضافه می​شود:

this.penetration = null;

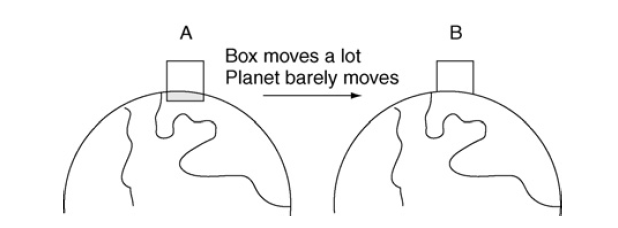
توجه داشته باشید که همانند سرعت نزدیک شدن، عمق نفوذ هم دارای بزرگی است و هم علامت. عمق نفوذ منفی بیانگر آن است که دو جسم هرگز از هم نمی​گذرند. عمق نفوذ صفر نشان‌دهنده‌ی این است که دو جسم فقط هم دیگر را لمس می​کنند.

به منظور تحلیل تداخل، تنها نیاز به کنترل عمق نفوذ داریم. در صورتی که مقدار عمق نفوذ صفر یا کمتر از صفر باشد، نیاز به انجام هیچ اقدامی​نیست. در غیر این صورت اجسام را فقط به اندازه​​ای می​توانیم از هم دور کنیم که عمق نفوذ برابر صفر شود. عمق نفوذ باید درجهت نرمال تماس داده شود. اگر اجسام را درجهت نرمال تماس به اندازه عمق نفوذ جابه جا کنیم، آنگاه دو جسم دیگر با هم در تماس نخواهند بود. زمانی که فقط یک جسم در تماس دخالت دارد (تداخل با صحنه بازی رخ می​دهد) نیز، همین روش انجام می​شود یعنی عمق نفوذ در جهت نرمال تماس خواهد بود.

حال که مقدار جابه‌جایی کل دو جسم (مقدار نفوذ) و جهتی که اجسام باید جابه‌جا شوند مشخص شده است، مقدار جابه‌جایی جداگانه هر یک از اجسام باید مشخص شود.

اگر تنها یک جسم در تماس درگیر باشد، راه حل ساده است: جسم باید به اندازه کل فاصله تداخل، جابه‌جا شود. اگر دو جسم وجود داشته باشد، دراین صورت طیف گسترده​​ای از راه حل​ها وجود دارد. برای مثال، می​توان به سادگی هر یک از دو جسم را به اندازه نصف مقدار نفوذ جابه جا کرد. این راه در برخی از وضعیت​ها قابل انجام است اما مشکل باورپذیری ایجاد می​کند. برای نمونه، جعبه​​ای کوچک را که بر روی سطح سیاره​​ای را شبیه‌سازی شده است در نظر بگیرید، اگر جعبه مقداری درداخل سیاره نفوذ کرده باشد، آیا جعبه و سیاره باید به اندازه مساوی جابه‌جا شوند ؟

باید به این نکته توجه کرد که درابتدا نفوذ چگونه به وجود آمده است و در واقعیت چه اتفاقی رخ می​دهد. شکل 15، حالت A، جعبه و سیاره را در حالت نفوذ نشان می​دهد. باید تا جایی که ممکن است به حالت B تصویر نزدیک شویم، به گونه​​ای که گویی فیزیک واقعی در جریان است.



شکل مشکل تداخل در شکل​های بزرگ

برای انجام این عمل، اجسام را به نسبت عکس جرمشان جابه جا می​کنیم. جسمی​با جرم زیاد بسیار کم جابه جا می​شود و جسمی​با جرم ناچیز باید به مقدار زیادی جایه جا شود. اگر یکی از دو جسم دارای جرم بی نهایت باشد، اصلا جابه جا نمی​شود و جرم دیگر باید تمام جابه‌جایی را انجام دهد.

جمع کل مقدار جابه‌جایی برابر مقدار تداخل است:

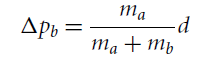


ap∆ برابر فاصله​​ای است که جسم a باید طی کند. این دو فاصله از طریق نسبتی که با جرم اجسام دارند به هم مرتبط هستند.

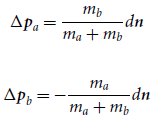


این دو رابطه اگر با هم ترکیب شوند معادلات زیر به دست می​آید:





با ترکیب این دو با جهت نرمال تماس، تغییر کل در جهت برداری به دست می​آید:



n نشان دهنده نرمال تماس است (علامت منفی موجود در رابطه دوم به دلیل این است که نرمال تماس از دید جسم a محاسبه شده است.).

پیاده​سازی معادله تحلیل تداخل به صورت زیر است:

ParticleContact.prototype.resolveInterpenetration = function(duration){

if (this.penetration <= 0) return;

var totalInverseMass = this.particle[0].inverseMass;

if (this.particle[1]){

totalInverseMass += this.particle[1].inverseMass;

}

if (totalInverseMass <= 0) return;

var movePerIMass = new Point(this.contactNormal.x \*(-this.penetration / totalInverseMass),this.contactNormal.y \*(-this.penetration / totalInverseMass));

this.particle[0].x = this.particle[0].x + movePerIMass.x \* this.particle[0].inverseMass;

this.particle[0].y = this.particle[0].y + movePerIMass.y \* this.particle[0].inverseMass;

if (this.particle[1])

{

this.particle[1].x = this.particle[1].x + movePerIMass.x \* this.particle[1].inverseMass;

this.particle[1].y = this.particle[1].y + movePerIMass.y \* this.particle[1].inverseMass;

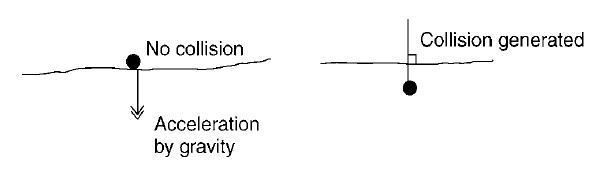
}

}

تا این جا پیاده​سازی انجام شده، تغییر سرعت انجام شده در زمان برخورد و تحلیل تداخل را اعمال می​کند. حال با اجرای سیستم تحلیل تماس ملاحظه می​شود که این سیستم، برای برخوردهای با سرعت متوسط به خوبی عمل می​کند. اما اجسام ایستا و ساکن (برای مثال ذره​​ای که برروی یک میز به صورت ساکن قرار دارد)، ممکن است شروع به لرزش کنند وحتی برخی اوقات به هوا برمی​خیزند. برای داشتن یک سیستم تحلیل تماس کامل و پایدار، نیاز به بازبینی اتفاقاتی که در زمان برخورد دو جسم با سرعت نزدیک شدن کم یا صفر رخ می​دهد، داریم.

در این موتور فیزیک، سیستم تحلیل تماس برای برخوردهای با سرعت متوسط طراحی شده است. شبیه‌ساز فیزیکی که در این جا ارائه شد، توانایی کنترل این شرایط را دارد. اما یابنده​های برخورد ممکن است در این شرایط نتایج عجیبی تولید کنند. برای مثال، ممکن است قبل از این که یابنده برخورد تشخیص دهد که دو جسم با هم تماس داشته اند، دو جسم از هم بگذرند. این امکان وجود دارد که در زمانی که یابنده برخورد بتواند برخورد را تشخیص دهد، دو جسم تا نیمه از هم گذشته باشند و در حال جدا شدن از هم باشند، در این شرایط اجسام دارای سرعت دور شدن مثبت هستند و هیچ ضربه​​ای ایجاد نمی​شود.

شرایط موجود در تصویر زیر را در نظر بگیرید.



شکل مشکل برخوردهای ایستا

ذره​​ای ساکن بر روی زمین قرار دارد. تنها نیرویی که به آن وارد می​شود نیروی جاذبه زمین است. در اولین فریم، ذره شتابی به سوی پایین می​گیرد. سرعت ذره افزایش می​یابد ولی موقعیت ذره ثابت می​ماند (در ابتدای فریم، ذره فاقد سرعت است.). در فریم دوم، موقعیت به روز می​شود و سرعت دوباره افزایش می​یابد. حال ذره در حال حرکت به سمت پایین است و در حال تداخل با زمین است. یابنده برخورد نفوذ را تشخیص می​دهد، تحلیلگر تماس ذره را مورد بررسی قرار می​دهد و متوجه می​شود که ذره دارای سرعت نفوذ زیر است:



با اعمال پاسخ برخورد، ذره سرعتی برابر



پیدا می​کند و از حالت تداخل خارج می​شود. در فریم سوم، ذره دارای سرعتی رو به بالا است که موجب می​شود جسم از زمین جدا شود و به هوا برود. سرعت رو به بالا بسیار کم است، اما به اندازه​​ای که است که می​توان متوجه حرکت آن شد. در واقع، اگر فریم اول یا دوم به طور غیر معمولی طولانی باشد، سرعت رو به بالا این امکان را دارد که به طرز قابل توجه​​ای افزایش یابد و ذره را به سرعت به آسمان بفرستد. اگر این الگوریتم برای بازی با نرخ فریم متغیر پیاده​سازی شود و به تدریج نرخ فریم را کاهش دهیم ( برای نمونه، با انجام کاری دیگر در زمینه تصویر)، هر جسم ساکنی به طور ناگهانی می​پرد.

برای حل این مشکل، دو روش وجود دارد. در روش اول، نیاز به تشخیص زود هنگام تماس داریم. در این مثال، پس از گذشت دو فریم، متوجه وجود مشکل شدیم. اگر یابنده برخورد را به گونه​​ای تنظیم کنیم که تماس​هایی که نزدیک هستند اما تداخل کمی​دارند را باز گرداند، می​توان بعد از گذشت فریم اول، تماس را مدیریت کرد.

در روش دوم، باید زمانی را که جسم دارای سرعتی است که تنها ناشی از نیرو​های وارد شده بر آن در یک فریم است، تشخیص دهیم. پس از اتمام فریم اول، سرعت جسم تنها ناشی از نیروی جاذبه وارد بر آن در یک فریم است. می​توان سرعت در انتهای فریم را به سادگی از طریق ضرب نیرو در طول مدت زمان فریم به دست آورد. اگر سرعت واقعی ذره، مساوی و یا کمتر از این مقدار باشد ( یا حتی مقدار ناچیزی بیشتر از این سرعت)، می​توان متوجه شد که ذره در فریم قبلی ساکن بوده است. در این حالت، تماس احتمالا تماسی ایستا خواهد بود نه تماس برخوردی و به جای به کار بردن محاسبات ضربه مربوط به برخورد، ضربه​​ای به ذره اعمال می​کنیم که موجب شود ذره سرعت دور شدنی برابر صفر به خود بگیرد.

به طور کلی در یک تماس ایستا، مسائل زیر رخ می​دهد: در ابتدا فرصتی برای افزایش سرعت دور شدن وجود نخواهد داشت، در نتیجه بعد از تماس، سرعت دور شدن برابر صفر خواهد بود. در مثال بالا، سرعت به روش به کار برده شده در تقسیم زمان به تعدادی فریم، وابسته است، در نتیجه می​توان با جسم به گونه​​ای رفتار کرد که گویی قبل از تماس دارای سرعت صفر بوده است. ذره​​ای که دارای سرعت صفر است و در مثال بالا ذره همواره در فریم اول باقی می​ماند.

از نقطه نظر دیگری نیز می​توان به این مسئله نگاه گرد. در واقع، در هر فریم، شاهد برخوردی با ضریب ارتجاع صفر هستیم. این مجموعه از برخورد​های جزیی، اجسام را ازهم جدا نگه می​دارد. از این جهت، به موتوری که قابلیت مدیریت تماس​های ایستا را دارد، "موتور برخورد جزیی" نیز گفته می​شود.

هنگامی​که دو جسم در وضعیت تماس ایستا قرار دارند، سرعت نسبی دو جسم از سرعت مطلق هر یک از دو جسم، از اهمیت بالاتری برخوردار است. دو جسم ممکن است با هم در یک جهت در تماس ایستا باشند، اما نسبت به هم، درجهت دیگر حرکت کنند. یک جعبه ممکن است بر روی زمین ساکن قرار داشته باشد، در حالی که ممکن است هم زمان بر روی سطح در حال لغزش باشد. در این جا به کدی به منظور مدیریت تماس​های لرزشی نیاز است که بتواند اجسامی​را که بر روی هم سر می​خورند، کنترل نماید. این بدین معنی است که دیگر امکان استفاده از سرعت مطلق هر جسم، به طور جداگانه وجود ندارد.

برای مدیریت این وضعیت، محاسبات مربوط به سرعت و شتاب تنها در جهت نرمال تماس صورت می​گیرد. ابتدا سرعت را در این جهت به دست می​آوریم، سپس بررسی می​کنیم که آیا این سرعت تنها توسط مولفه شتاب درجهتی یکسان ایجاد شده است یا خیر. اگر این گونه باشد، سرعت تغییر می​کند، بنابراین سرعت دور شونده یا نزدیک شونده در این جهت وجود ندارد. اما هنوز این امکان وجود دارد که سرعتی نسبی در جهات دیگر موجود باشد، با این وجود از آن صرف نظر می​شود.

پیاده​سازی این مورد خاص، به صورت زیر به تابع پردازش برخورد اضافه می​شود:

var accCausedVelocity = this.particle[0].acceleration;

if (this.particle[1]){

accCausedVelocity.x -= this.particle[1].acceleration.x;

accCausedVelocity.y -= this.particle[1].acceleration.y;

}

var accCausedSepVelocity = accCausedVelocity.scalarProduct(this.contactNormal) \* duration;

if (accCausedSepVelocity < 0)

{

newSepVelocity += this.restitution \* accCausedSepVelocity;

if (newSepVelocity < 0) newSepVelocity = 0;

}

به منظور نگه داشتن دو جسم در وضعیت تماس ایستا، در هر فریم تغییر کمی​در سرعت اعمال می​شود. این تغییر به منظور تصحیح افزایش سرعتی که از فرو رفتن دو جسم در هم در طول یک فریم به وجود می​آید، صورت می​گیرد.

روش برخورد جزیی که در بالا به آن اشاره شد، تنها یکی از روش​های ممکن در مدیریت تماس ایستا است. تماس ایستا یکی از دو چالش موجود در طراحی یک موتور فیزیک است ( چالش دیگر اصطکاک است که در واقع این دو چالش، با هم مطرح می​شوند.). روش​های حمله، پیچیدگی​ها و بی ثباتی​های بسیاری وجود دارد.

یک روش فیزیکی واقع گرایانه، توجه به این نکته است که در واقعیت نیرویی از جانب زمین بر ذره اعمال می​شود. این نیروی واکنش، جسم را به عقب فشار می​دهد تا شتاب کل جسم در جهت عمودی برابر صفر شود. جسم هر چقدر به سمت پایین نیرو وارد کند، زمین به همان میزان در جهت بالا نیرو وارد می​کند. می​توان چنین مولد نیرویی ایجاد کرد تا اطمینان حاصل شود که هیچ شتابی به سمت زمین وجود ندارد.

این روش در مورد ذره​هایی که فقط یک تماس با زمین دارند به خوبی عمل می​کند. اما وضعیت دررابطه با اجسام صلب و سخت پیچیده‌تر خواهد بود. این امکان وجود دارد که چندین نقطه تماس بین جسم و نقاط ساکن غیر قابل حرکت وجود داشته باشد. این امکان وجود ندارد که به صورت آنی نیروهای واکنش را در هر نقطه تماس محاسبه کرد به گونه​​ای که حرکت کلی جسم صحیح باشد.

تحلیلگر تماس، لیستی ازتماس​ها را از سیستم تشخیص برخورد، دریافت می​کند و اجسام در حال شبیه‌سازی را به روز می​کند تا بتواند تماس​ها را کنترل نماید. برای انجام این به روز رسانی، سه دسته کد وجود دارد:

١ـ تابع تحلیل برخورد که به اجسام، ضربه اعمال می​کند تا دور شدن آن​ها را شبیه‌سازی کند.

٢ـ تابع تحلیل تداخل که اجسام را از هم دور می​کند تا جایی که دیگر با هم تداخل نداشته باشند.

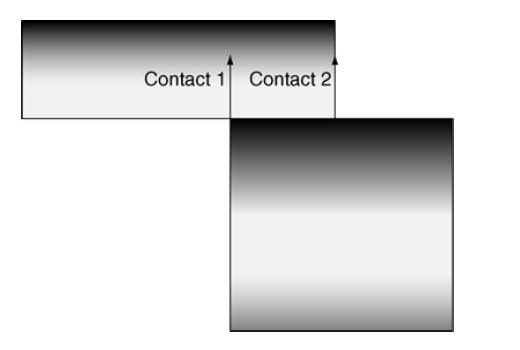
٣ـ کد مربوط به تماس ایستا که در داخل تابع تحلیل برخورد قرار دارد و تماس​هایی را مدیریت می​کند که بیشتر احتمال دارد ایستا باشند و نه برخورد.

این مسئله که کدام تابع نیاز به فراخوانی تماس دارد، بستگی به سرعت دور شدن و مقدار تداخل دارد. تحلیل تداخل تنها زمانی انجام می​شود که تماس دارای عمق نفوذی با مقدار بزرگتر از صفر باشد. همچنین، در صورتی که دو جسم در هم نفوذ کرده باشند اما جدا از هم باشند، ممکن است تنها نیاز به اجرای تحلیل تداخل باشد و نیازی به تحلیل برخورد نباشد.

با صرف نظر از ترکیب توابع مورد نیاز، هر تماسی به طور جداگانه تحلیل می​شود. این عمل با هدف ساده کردن دنیای واقعی انجام می​شود. برخی از تماس​ها، به صورت سلسله مراتبی اثر می​کنند، اما برخی دیگر، ترکیب می​شوند و همزمان روی جسم اثر می​کنند. برخی همین روند را تکرار می​کنند، تماس​های متوالی را به ترتیب و تماس​های ایستا را به صورت همزمان تحلیل می​کنند.

در پیاده​سازی این موتور جستجو، نگاهی ساده به مسائل شده است و در پایان هر فریم تمام تماس​ها به طور همزمان تحلیل نشده است. با این وجود، برای رسیدن به نتایج دقیق تر، نیاز به تحلیل تماس​ها به ترتیب صحیح است.

اگر جسمی، همانگونه که در تصویر زیر نیز نشان داده شده است، دارای دو تماس همزمان باشد، تغییر سرعت آن جسم به منظور تحلیل یکی از تماس​ها، ممکن است سرعت دور شدن در تماس دیگر را تغییر دهد. در این تصویر، اگر اولین تماس تحلیل شود، دومین تماس دیگر برخورد تلقی نخواهد شد، یعنی جدا می​شود. اما اگرابتدا تماس دوم تحلیل شود، تماس اول همچنان نیاز به تحلیل شدن دارد و تغییر سرعت کافی نخواهد بود.



شکل مشکل اولویت تداخل​ها

به منظور جلوگیری از این چنین اعمال غیر ضروری، ابتدا شدیدترین و سخت​ترین تماس را باید تحلیل کرد: یعنی تماسی که کمترین سرعت دورشدن (منفی​ترین) را دارد. علاوه بر راحت بودن، این عمل بهترین عمل فیزیکی واقع گرایانه قابل انجام خواهد بود. در تصویر، اگر رفتار وضعیتی که هر سه جسم درآن قرار دارند را با رفتار وضعیتی که یکی از دو بلوک زیری برداشته شده است مقایسه کنیم، نتیجه نهایی بسیار مشابه حالتی است که بلوک A وجود دارد اما بلوک B وجود نداشته باشد. به عبارت دیگر، سخت​ترین برخوردها، رفتار شبیه‌سازی را تحت سلطه خود دارند. اگر برای مدیریت برخوردها نیاز به اولویت بندی باشد، اولویت با واقعی​ترین برخورد خواهد بود. تصویر بالا نشان دهنده پیچیدگی الگوریتم تحلیل تماس است. اگر یکی از برخوردها مدیریت شود، سرعت دور شدن تماس​های دیگر امکان دارد تغییر یابد. نمی​توان تنها با مرتب کردن تماس​ها بر اساس سرعت دور شدنشان، به ترتیب آن​ها را تحلیل کرد. چرا که با تحلیل اولین برخورد، ممکن است سرعت دور شدن تماس بعدی مثبت شود و دیگر نیازی به پردازش نباشد.

مشکل ظریف دیگری نیز وجود دارد که در بسیاری از وضعیت​ها رخ نخواهد داد. برای مثال، ممکن است با وضعیتی رو به رو شویم که تحلیل تماس دوم، تماس اول را دوباره به برخورد بازگرداند، در نتیجه نیاز به تحلیل دوباره تماس اول داریم. خوشبختانه، می​توان نشان داد که در برخی از انواع شبیه‌سازی (به خصوص شبیه​سازی​های فاقد اصطکاک)، این حلقه تدریجا به جواب صحیح ختم می​شود و نیازی به ادامه حلقه تا ابد نیست و در پایان با وضعیتی که اصلاحات تا اندازه​​ای بزرگ شود که شبیه‌سازی منفجر شود، مواجه نخواهیم شد. اما این روش ممکن است مدت زیادی زمان ببرد و هیچ راهی برای تخمین زمان لازم وجود ندارد. به همین دلیل، محدودیتی بر روی تعداد تحلیل​هایی که در هر فریم می​تواند انجام شود، اعمال می​کنیم.

تحلیلگر تماس مورد استفاده در این پروژه، از الگوریتم زیر استفاده می​کند:

١ـ محاسبه سرعت دور شدن هر تماس و مشخص کردن تماسی با کمترین (منفی​ترین) مقدار.

٢ـ اگر کم​ترین سرعت دور شدن، بزرگ‌تر و یا مساوی صفر باشد، از الگوریتم خارج می​شویم.

٣ـ پردازش الگوریتم پاسخ برخورد برای تماسی با کمترین سرعت دور شدن.

٤ـ اگر تکرار​های بیشتری وجود دارد، الگوریتم به گام اول باز می​گردد.

الگوریتم به صورت خودکار، تمام تماس​هایی را که قبلا تحلیل شده اند، دوباره بررسی می​کند و تماس​هایی را که در حال جدا شدن هستند، نادیده می​گیرد. در هر تکرار، سخت​ترین برخورد تحلیل می​شود.

تعداد دفعات تکرار، باید حداقل برابر تعداد تماس​ها باشد ( تا هر کدام حداقل یک بار فرصت بررسی شدن را داشته باشند.). در شبیه​سازی​های ساده، تعداد تکراری برابر تعداد تماس​ها به خوبی عمل می​کند. در این جا، تعداد تکرار​ها به طور تخمینی، دو برابر تعداد تماس​ها در نظر گرفته شده است. اما در مورد تماس​های پیچیده، یقینا به تعداد بیشتری تکرار نیاز است. همچنین می​توان روی الگوریتم، هیچ محدودیتی قرار نداد تا بتوان نحوه عملکرد آن را بررسی کرد.

تا این جا، از تداخل چشم پوشی کردیم. می​توان تحلیل تداخل را با تحلیل برخورد ترکیب کرد. البته راه حل بهتر، جداسازی این دو تحلیل در دو فاز مجزا است. ابتدا، برخوردها را با استفاده از الگوریتم قبلی، به ترتیب تحلیل می​کنیم. سپس، تداخل​ها را تحلیل می​کنیم.

جداسازی دو گام تحلیل، این امکان را فراهم می​آورد که از ترتیبی متفاوت برای تحلیل تداخل استفاده کرد و بر اساس سرعت، تماس​ها را مرتب نکرد. در این جا نیز، نیاز به نتایج واقع گرایانه داریم. می​توان تماس​ها را براساس شدت و سختی مرتب کرد و سپس تحلیل را انجام داد. اگر این دو مرحله را با هم ادغام کنیم، ترتیب برای یکی از دو تحلیل، ترتیب مطلوبی نخواهد بود.

تحلیل تداخل، ازالگوریتمی​ مشابه الگوریتم تحلیل برخورد استفاده می​کند. همانند قبل، در هر تکرار، نیاز به محاسبه دوباره تمامی​مقادیر نفوذ داریم. این نکته حائز اهمیت است که مقادیر تداخل توسط یابنده برخورد فراهم می​شود. از آن جایی که اجرای الگوریتم تشخیص برخورد زمان بر است، نمی​توان در پایان هر تکرار آن را اجرا کرد. در نتیجه به منظور به روز رسانی مقادیر تداخل، مقدار جابه‌جایی دو جسم در تکرار قبلی را ذخیره می​کنیم. سپس دو جسم در هر تماس بررسی می​شوند، اگر یکی از دو جسم و یا هر دو، در فریم قبلی جابه جا شده باشد، مقدار تداخل با یافتن مولفه حرکت در جهت نرمال تماس، به روز می​شود.

با در نظر گرفتن تمام این موارد، تابع تحلیلگر تماس به صورت زیر خواهد بود:

function ParticleContactResolver(iterations)

{

this.iterations = iterations;

this.iterationsUsed = 0;

}

ParticleContactResolver.prototype.resolveContacts = function(contactArray,numContacts,duration){

this.iterationsUsed = 0;

while(this.iterationsUsed < this.iterations)

{

var max = 0;

var maxIndex = numContacts;

for (var i = 0; i < numContacts; i++)

{

var sepVel = contactArray[i].calculateSeparatingVelocity();

if (sepVel < max)

{

max = sepVel;

maxIndex = i;

}

}

contactArray[maxIndex].resolve(duration);

this.iterationsUsed++;

}

}

ممکن است تعداد دفعات تکرار به کار برده شده در تحلیل تداخل با تعداد دفعات تکرار استفاده شده در تحلیل برخورد​ها یکسان نباشد. می​توان پیاده​سازی تابع را به گونه​​ای انجام داد که هر تحلیل، تعداد دفعات تکرار خاص خود را داشته باشد.

در عمل، نیازی به استفاده از دو مقدار متفاوت نیست. هر چه شبیه‌سازی دشوار‌تر شود و اجسامی​در حال تعامل وجود داشته باشد، تعداد دفعات تکرار تحلیل برخورد، تقریبا به میزان افزایش تعداد دفعات تحلیل تداخل، افزایش می​یابد. در پیاده​سازی انجام شده در بالا، برای هر دو تحلیل، تعداد تکرار یکسان در نظر گرفته شده است.

محاسبه مجدد سرعت نزدیک شدن و مقدار تداخل، زمان برترین قسمت این الگوریتم است. در صورت وجود تعداد تماس زیاد، سرعت اجرای موتور فیزیک، تحت شعاع قرار می​گیرد. در عمل، از آن جایی که ممکن است یک تماس بر تماس​های دیگر اثری نداشته باشد، بسیاری از به روز رسانی​ها بی فایده خواهند بود.

رویکرد دیگری در ایجاد یک موتور فیزیک قابل استفاده است. این رویکرد از لزوم تحلیل تداخل با ایجاد ترتیبی برای تحلیل تماس​ها، اجتناب می​کند. می​توان به جای به روز رسانی موتور فیزیک در هر فریم، به روز رسانی​های متعددی بر مبنای برخورد​ها انجام داد:

این شیوه به صورت زیر است:

در صورت عدم وجود برخورد، اجسام بر اساس قوانین حرکت و مولدهای نیرو، آزادانه به اطراف حرکت می​کنند.

زمانی که برخوردی روی می​دهد. نقطه برخورد، دقیقا در نقطه​​ای است که دو جسم با هم تماس پیدا می​کنند. در این مرحله، هیچ تداخلی وجود ندارد.

اگر بتوان زمان دقیق برخورد را پیش بینی کرد، ابتدا از قوانین معمول حرکت استفاده می​کنیم، سپس محاسبات مربوط به ضربه را انجام می​دهیم و دوباره قوانین مربوط به حرکت نرمال را آغاز می​کنیم.

اگر برخورد​های متعددی رخ دهد، این برخورد​ها را به ترتیب پردازش می​کنیم. در زمان بین دو برخورد، با استفاده از قوانین نرمال حرکت، تمام اجسام را به روز رسانی می​کنیم.

در واقع، این نوع موتور، از الگوریتم زیر پیروی می​کند:

١ـ زمان شروع را برابر زمان کنونی شبیه‌سازی و زمان پایان را برابر پایان زمان تقاضای به روز رسانی فعلی قرار می​دهیم.

٢ـ به روز رسانی کاملی برای کل فاصله زمانی انجام می​دهیم.

٣ـ یابنده برخورد را اجرا می​کنیم و لیست برخوردها را جمع آوری می​کنیم.

٤ـ اگر هیچ برخوردی صورت نگرفته باشد، از الگوریتم خارج می​شویم.

٥ـ برای هر برخورد، زمان دقیق اولین برخورد محاسبه می​شود.

٦ـ اولین برخوردی که باید صورت گیرد، انتخاب می​شود.

٧ـ اگر اولین برخورد پس از زمان پایان رخ دهد، از الگوریتم خارج می​شویم.

٨- به روز رسانی کامل مرحله دو، حذف می​شود و به روز رسانی دیگری از زمان شروع تا زمان اولین برخورد انجام می​شود.

٩ـ برخورد با اعمال ضربه​های مناسب پردازش می​شود (به تحلیل تداخل نیازی نیست چرا که در زمان برخورد دو جسم فقط در تماس با هم هستند.).

١٠ـ زمان شروع را برابر زمان اولین برخورد قرار می​دهیم، زمان پایانی را تغییر نمی​دهیم و به گام اول باز می​گردیم.

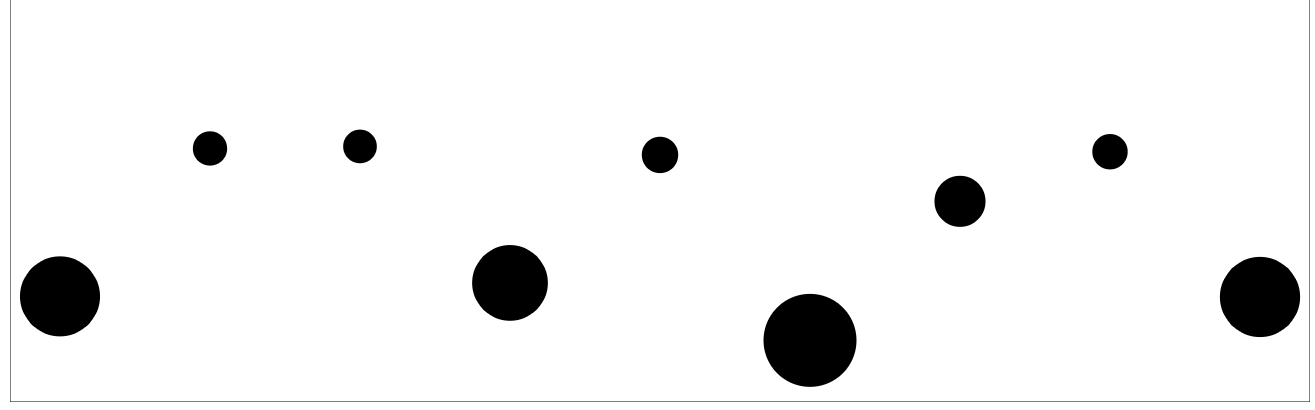
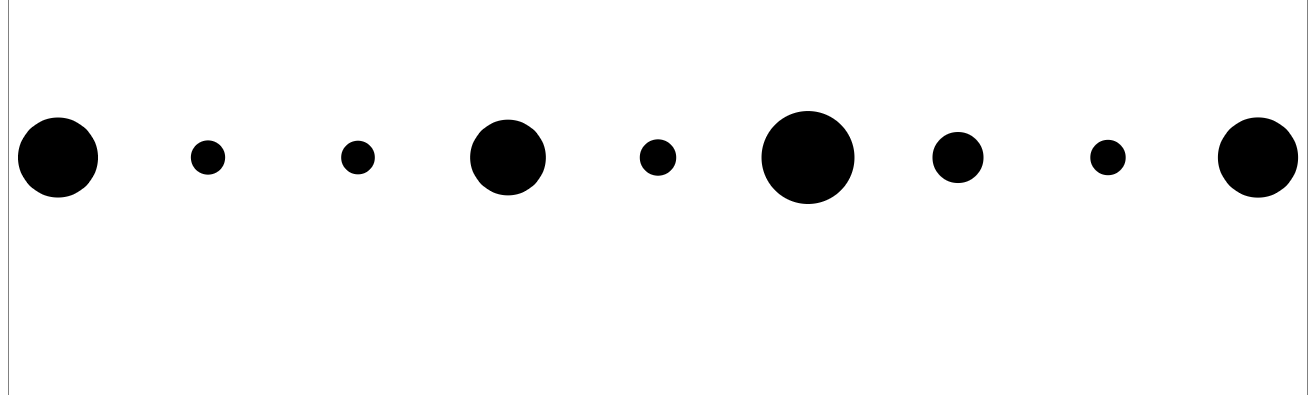
این الگوریتم، نتیجه دقیقی باز می​گرداند و مشکلات مربوط به تحلیل تداخل را ندارد. این الگوریتم معمولا در برنامه​های مهندسی فیزیک که دقت از اهمیت بالایی برخوردار است، به کار می​رود. اما متاسفانه، این الگوریتم بسیار زمان بر است.

برای هر برخورد، یابنده تشخیص، اجرا می​شود. همچنین، هر بار به روز رسانی فیزیک دوباره اجرا می​شود. اما هنوز هم نیاز به کدی داریم که مدیریت تماس​هاس ایستا را انجام دهد. در غیر این صورت، در هر تکرار، تماس ایستا به عنوان اولین برخورد، بازگردانده می​شود. حتی اگر تماس ایستا موجود نباشد، خطاهای عددی در محاسبات مربوط به تشخیص برخورد، می​تواند دوره​های بدون پایان ایجاد کند.

در پروژه​های مربوط به بازی، این روش عملی نیست. به روز رسانی در هر فریم، راه حل بهتری است، زیرا تمام تماس​ها از نظر تداخل و سرعت تحلیل می​شوند. تنها بازی​هایی که امکان استفاده از این روش دارند، بازی​های بیلیارد و اسنوکر هستند. در این بازی​ها ترتیب برخوردها و موقعیت توپ​ها در زمان برخورد بسیار مهم است. در شبیه​سازی​های با اهمیت، باید از الگوریتم قبلی استفاده کرد.

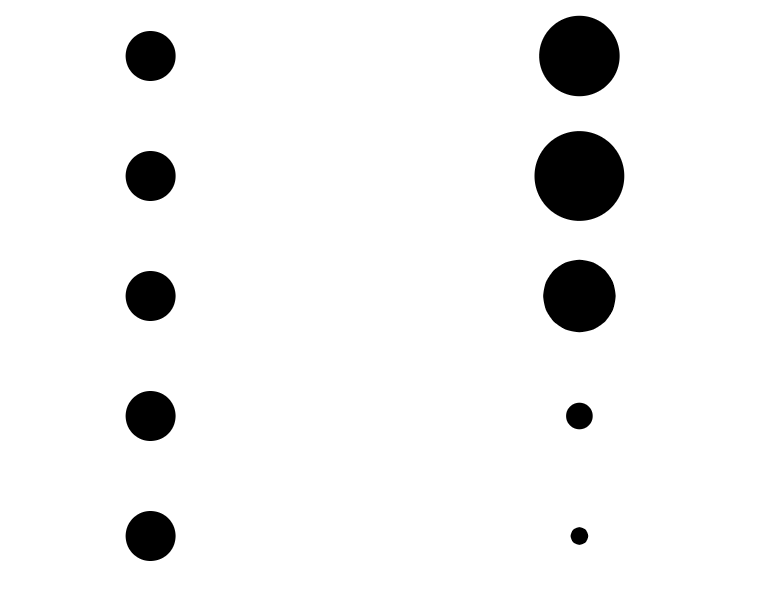
در تصاویر زیر مثال​هایی از اعمال محدودیت​های قوی را می بینیم.

در مثال زیر اثر جنس جسم و در واقع ضریب ارتجاع بر سرعت و رفتار جسم با ضریب​های متفاوت را می بینیم. همان طور که مشاهده می شود جسم با ضریب بالاتر (در این جا با اندازه کوچتر) ارتجاع بالاتری دارد:

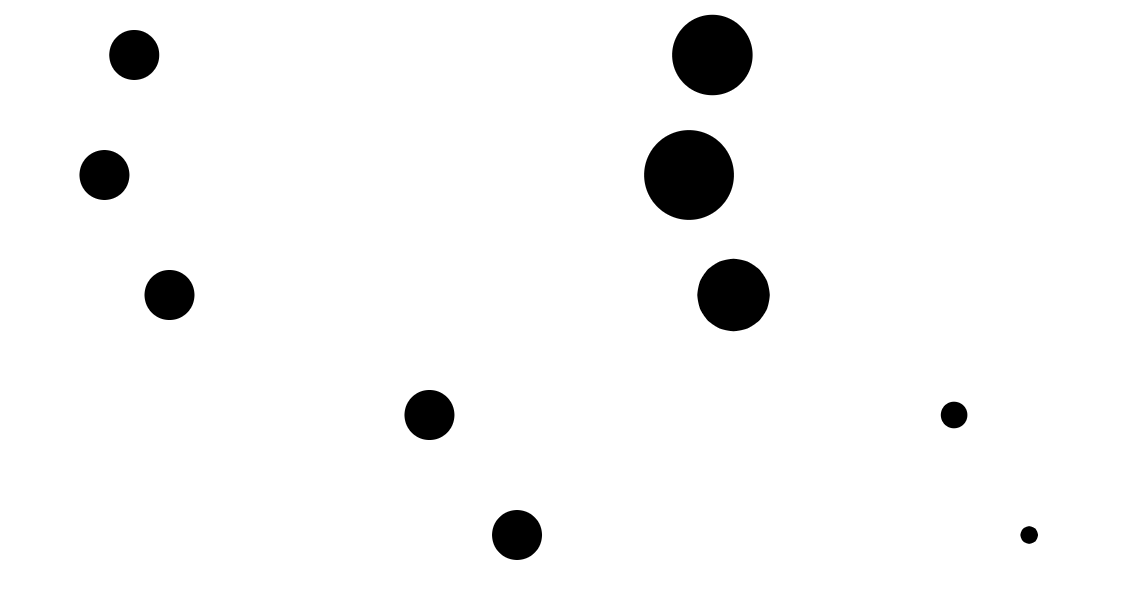


شکل تحلیل برخورد با ثابت متفاوت

در یک مثال دیگر اثر جرم جسم بر سرعت بعد از برخورد جسم مشاهده می شود:



شکل برخورد با جرم​های متفاوت



شکل بعد از برخورد

## پدیده​های برخوردگونه

همان طور که در مورد فنر دیدیم، به چند نوع از اتصالات که با استفاده از تکنیک​های مطرح شده، قابل مدل کردن هستند، نگاهی می​اندازیم.

می​توان برخورد را تلاشی برای جدا نگه داشتن دو جسم در فاصله​​ای بسیار کم، تصور کرد. تماس زمانی بین دوجسم ایجاد می​شود که دو جسم به شدت به هم نزدیک شوند. می​توان از تماس برای نگه داشتن دو جسم در کنار هم استفاده کرد.

کابل محدودیتی است که دو جسم را مجبور می​کند به اندازه طول کابل از هم فاصله داشته باشند. اگر از کابلی ضعیف استفاده شود، دو جسم تا زمانی که به هم نزدیک هستند، اثر کابل را حس نمی​کنند. اما اگر کابل محکم کشیده شود، دو جسم بیشتر از طول کابل از هم دور نخواهند شد. بسته به خصوصیات و ویژگی​های کابل، اجسام ممکن است کمی​بیش‌تر یا کم‌تر از این طول، به همان شیوه‌ی اجسام در حال برخورد، جابه جا شوند. کابل دارای یک ضریب ارتجاع است که مقدار جابه‌جایی را کنترل می​کند.

کابل را می​توان با ایجاد تماس در زمانی که دو انتهای آن به مقدار زیاد کشیده می​شود، مدل کرد. این تماس بسیار شبیه تماس​های استفاده شده در رابطه با برخورد​ها است، با این تفاوت که نرمال تماس عکس می​شود. نرمال تماس در این حالت، به اجسام در جهت نزدیک شدن فشار می​آورد، نه در جهت دور شدن. مقدار تداخل تماس هم به مقداری که کابل بیش‌تر از حد خود کشیده شود، مرتبط است.

پیاده​سازی مولد تماس برای کابل به صورت زیر است:

function ParticleLink()

{

this.particle = new Array();

}

ParticleLink.prototype.currentLength = function(){

var relativePos = new Point(this.particle[0].position.x - this.particle[1].position.x,this.particle[0].position.y - this.particle[1].position.y);

return relativePos.length;

}

ParticleContact.prototype.fillContact = function(contact,limit){

return 0;

}

function ParticleCable()

{

ParticleLink.call(this);

this.maxLength = null;

this.restitution = null;

}

ParticleCable.prototype = new ParticleLink();

ParticleCable.prototype.constructor = ParticleCable;

ParticleCable.prototype.fillContact = function(contact, limit){

var length = this.currentLength();

if (length < this.maxLength)

{

return 0;

}

contact.particle[0] = this.particle[0];

contact.particle[1] = this.particle[1];

var normal = new Point(this.particle[1].position.x - this.particle[0].position.x,this.particle[1].position.y - this.particle[0].position.y);

normal = normal.normalize();

contact.contactNormal = normal;

contact.penetration = length-this.maxLength;

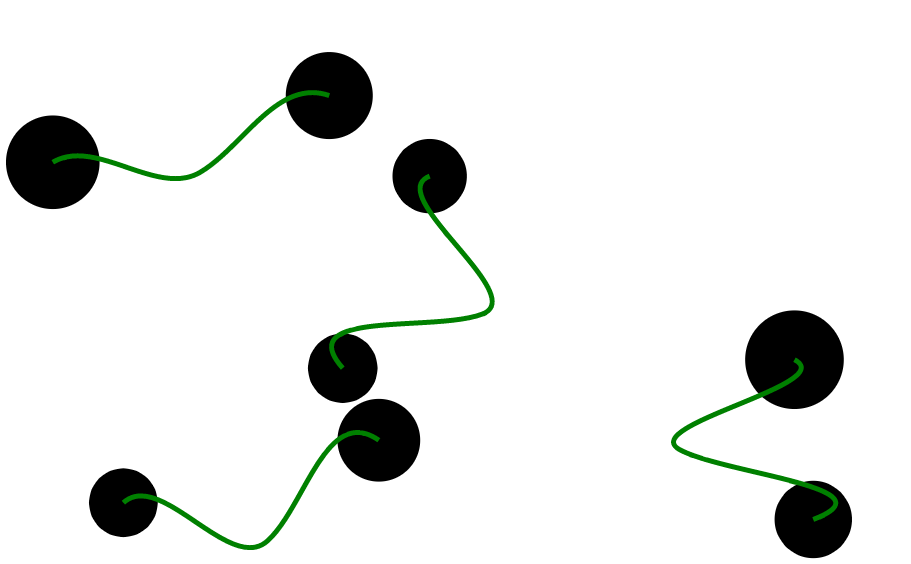
contact.restitution = this.restitution;

return 1;

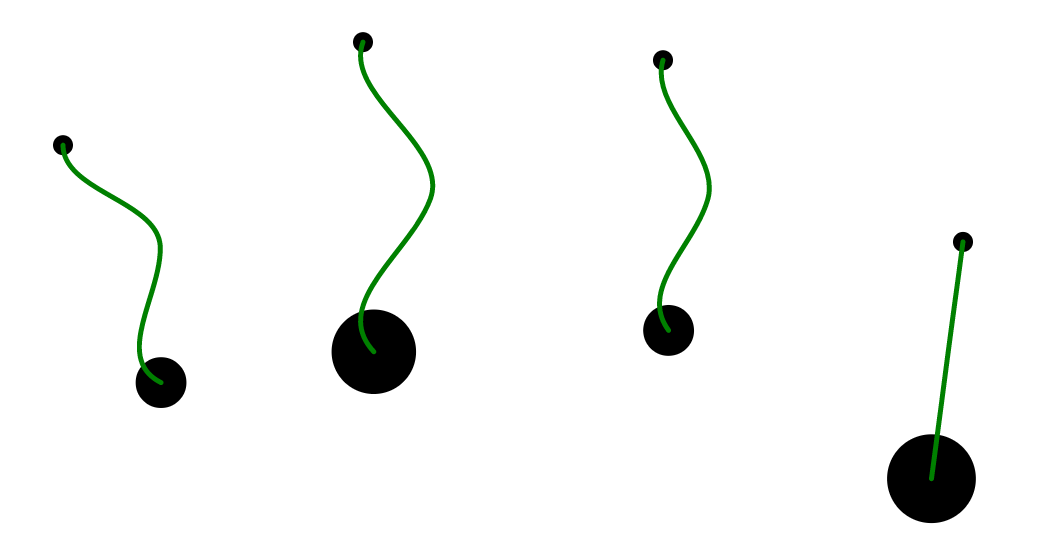
}

کد بالا به عنوان یک یابنده برخورد عمل می​کند: حالت فعلی کابل را بررسی می​کند و اگر کابل به حد انتهایی خود رسیده باشد، یک تماس برمی​گرداند. این تماس به تماس​های دیگری که توسط یابنده برخورد ایجاد شده است، اضافه می​شود و توسط الگوریتم تحلیلگر نرمال تماس پردازش می​شود.

در تصاویر زیر مثالی از پیاده​سازی چند کابل و ذرات متصل به آن​هار امی توانید مشاهده کنید:



شکل مثال اول طناب‌ها



شکل مثال دوم طناب‌ها

میله رفتار کابل و برخورد را ترکیب می​کند. دو جسمی​که توسط میله به هم وصل شده اند، نه می​توانند به هم نزدیک شوند و نه از هم دور شوند. دو جسم، در یک فاصله‌ی مشخصی از هم قرار دارند. پیاده​سازی میله مشابه پیاده​سازی مولد تماس کابل خواهد بود. در هر فریم، حالت فعلی میله بررسی می​شود و تماسی ایجاد می​شود که دو انتها را یا به هم نزدیک می​کند و یا دو انتها را ازهم دور می​کند.

در این جا نیاز به تغییر دو مورد که قبلا به آن اشاره شد، است. اول این که، ضریب ارتجاع همیشه برابر صفر خواهد بود. زیرا دو انتها در هیچ جهتی حرکت و لرزش ندارند. از آن جایی که این دو در فاصله‌ی ثابتی از هم قرار می​گیرند، سرعت نسبی بین آن​ها برابر صفر است. دوم این که، اگر در هر فریم، تنها یکی از دو تماس (دور شدن یا نزدیک شدن) را اعمال کنیم، میله به لرزش در خواهد آمد. در فریم​های متوالی، این امکان وجود دارد که ابتدا میله کوتاه و سپس بلند شود و هر تماس میله را به عقب یا جلو بکشاند. برای جلوگیری از وقوع این حالت، در هر فریم، هر دو تماس را ایجاد می​کنیم. اگر به یکی از تماس​ها نیاز نباشد (مثلا، سرعت دور شدن بزرگتر از صفر باشد یا تداخل وجود نداشته باشد.)، از آن چشم‌پوشی می​شود. وجود تماس اضافی، به الگوریتم تحلیلگر تماس کمک می​کند تا میله پایدارتر شود.

این روش در زمانی که مجموعه​​ای پیچیده از میله​ها وجود داشته باشد، بسیار ضعیف عمل می​کند، چراکه تعداد دفعات تکرار مورد نیاز، برای رسیدن به حالتی پایدار به شدت افزایش می​یابد. اگر تعداد دفعات تکرار کم باشد، لرزش دوباره مشاهده خواهد شد.

پیاده​سازی مولد تماس به صورت زیر خواهد بود:

ParticleRod.prototype.fillContact = function(contact,limit){

var currentLen = this.currentLength();

if (currentLen == this.length)

{

return 0;

}

contact.particle[0] = this.particle[0];

contact.particle[1] = this.particle[1];

var normal = new Point(this.particle[1].position.x - this.particle[0].position.x,this.particle[1].position.y - this.particle[0].position.y);

normal = normal.normalize();

if (currentLen > this.length) {

contact.contactNormal = normal;

contact.penetration = currentLen - this.length;

} else {

contact.contactNormal = new Point(normal.x \* -1,normal.y \*-1);

contact.penetration = this.length - currentLen;

}

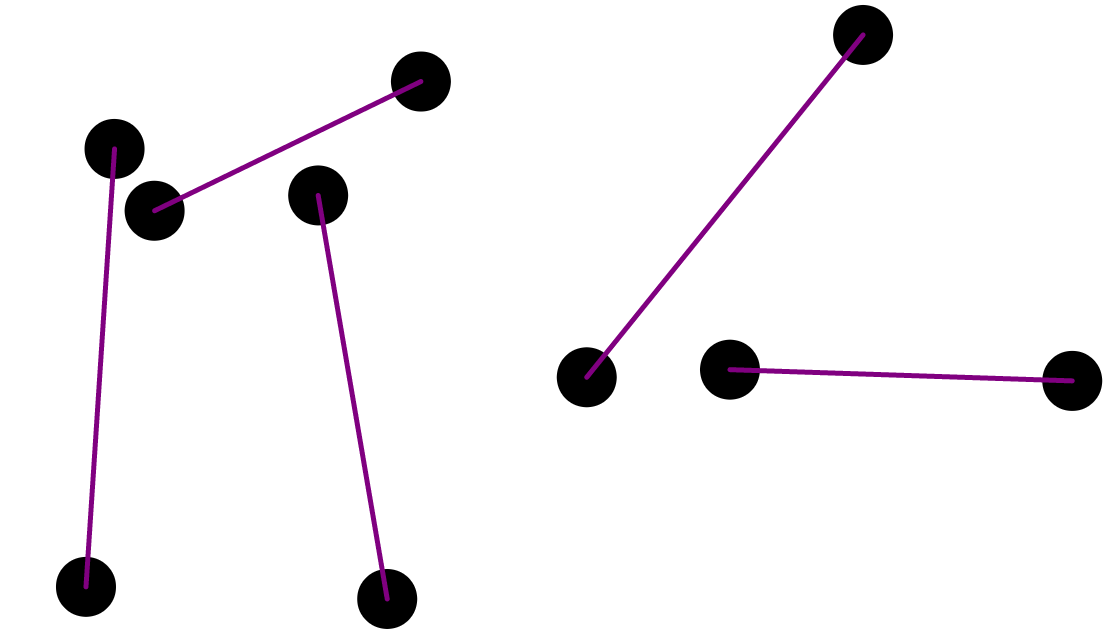
contact.restitution = 0;

return 1;

}

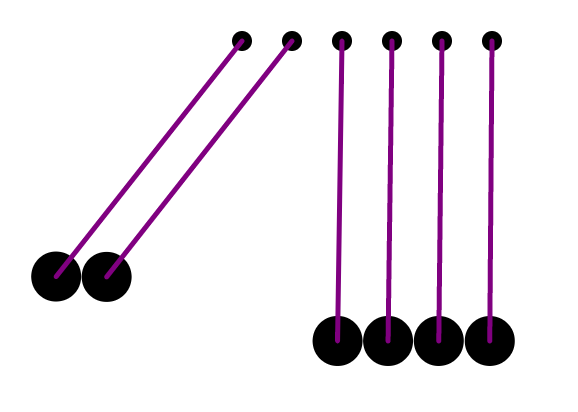
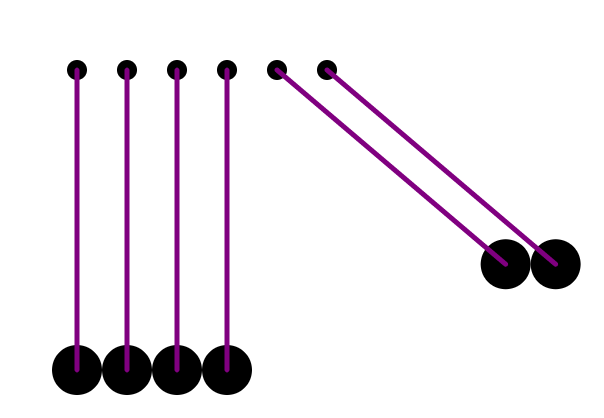
کد بالا، همواره دو تماس تولید می​کند که باید به لیست تماس​های برگردانده شده توسط یابنده برخورد اضافه شود و به تحلیلگر تماس تحویل داده شود.

در تصویر زیر نمایشی از یک مثال میله​ها آمده است:



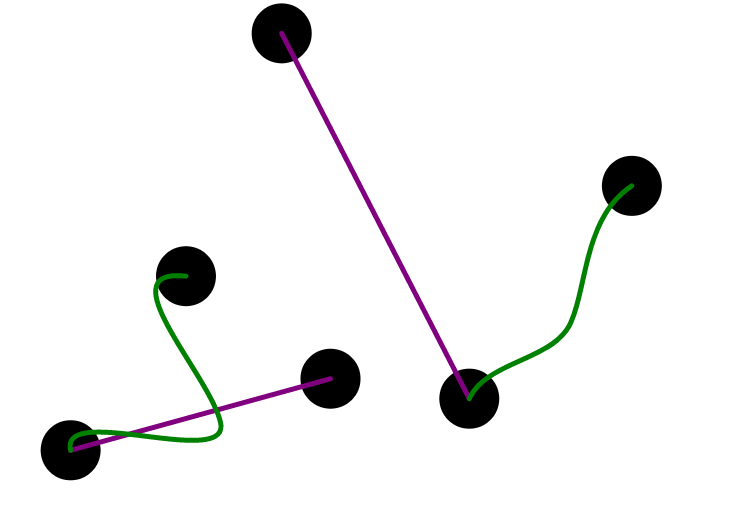
شکل مثال میله​ها

همچنین یک مثال کلاسیک از حرکت آونگ با کمک میله​ها پیاده​سازی شده که تصویر آن را در زیر مشاهده می کنید:



شکل مثال آونگ

یک مثال ترکیبی از میله​ها و کابل​ها را هم تصویر زیر آورده شده است:



شکل مثال ترکیبی

## جمع​بندی و نتیجه​گیری

در این فصل آخرین قسمت از موتور فیزیک یعنی محدودیت​های قوی را بررسی کردیم و نشان دایدم که چطور برخورد​ها و پدیده​های شبیه برخورد نقش حیاتی در ایجاد موتور تجمیع اجرام داشته باشند. انواع تولید برخورد​ها و تحلیل آن​ها حالا در موتور انجام می شود و همانطور که در فصل بعد خواهیم دید به راحتی می توان از آن​ها استفاده کرد تا یک موتور تجمیع کامل را در اختیار گرفت.

# فصل نهم: استفاده از موتور فیزیک تجمیع جرم

## مقدمه

تا این جا، موتور فیزیک تجمیع اجرامی​ساخته شد که توانایی شبیه‌سازی ذره و ایجاد اجسامی​از اشیا متعددی که توسط میله، کابل و فنر به هم متصل شده اند،تشکیل شده اند را دارد. حال زمان آن رسیده که این موتور را بر روی چند سناریو آزمایش کنیم. موتور پیاده​سازی شده، هنوز محدودیت​هایی دارد، برای نمونه، این موتور توانایی توصیف نحوه چرخش اجسام را ندارد. با استفاده از تجمیع اجرام، چرخش اجسام را به شکل ساختگی ایجاد می​کنیم. این تکنیک، در مورد برخی از برنامه​ها بسیار کارآمد است و دیگر نیازی به استفاده از موتور​های پیشرفته، نخواهد بود.

## نمای کلی موتور

موتور فیزیک تجمیع اجرام دارای سه جزء است:

١ـ خود ذرات، موقعیت، حرکت و جرم خود را ثبت می​کنند. برای ایجاد یک شبیه​سازی، باید بدانیم چه ذراتی نیاز است و سرعت مکان اولیه آن​ها را نیز مشخص کنیم. همچنین جرم معکوس این ذرات نیز باید مقدار دهی شود. شتاب جسم که ناشی از جاذبه زمین است، در جسم صلب نگه داری می​شود (می​توان آن را حذف کرد و با نیرویی جایگزین کرد.).

٢ـ مولد​های نیرو برای نگه داری و ثبت نیروهایی که در طول فریم​های متوالی اعمال می​شوند، به کار می​رود.

٣ـ سیستم برخورد، مجموعه‌ی تماس​های جسم را جمع آوری می​کند و آن را به تحلیلگر برخورد تحویل می​دهد. در این موتور فیزیک، دو مولد تماس در نظر گرفته شده است: یابنده برخورد و فشار کابل یا میله.

در هر فریم، اطلاعات داخلی تمام ذرات را محاسبه می​کنیم، مولدهای نیروی ذرات را فراخوانی می​کنیم. سپس integrator به منظور به روز کردن موقعیت و سرعت ذرات فراخوانی می​شود. در انتها، تمام تماس​های ذره جمع آوری می​شود و به تحلیلگر برخورد تحویل داده می​شود.

به منظور آسان‌تر کردن این فرایند، ساختاری ساده ایجاد می​کنیم که توانایی نگه داری هر تعداد جسم صلب را داشته باشد. این اجسام صلب در یک list linked به همان شیوه به کار رفته در مورد مولد​های نیرو، نگه داری می​شوند. این list linked در یک کلاس World قرار دارد که نمایانگر تمام دنیای شبیه‌سازی شده فیزیکی است:

function ParticleRegistration()

{

this.particle = null;

this.next = null;

}

function ParticleWorld(maxContacts,iterations)

{

this.firstParticle = null;

}

At each frame the startFrame method is first called, which sets up each object

ready for the force accumulation:

ParticleWorld.prototype.startFrame = function(){

var reg = this.firstParticle;

while (reg)

{

reg.particle.forceAccum.x = 0;

reg.particle.forceAccum.y = 0;

reg = reg.next;

}

}

باقی نیروها می توانند بعد از فراخوانی این تابع اعمال شوند.

همچنین سیستم دیگری برای نگهداری برخوردها داریم. همان طور که برای مولدهای نیرو هم همین کار را کردیم از رابط​های پلی مورفیک برای یابنده​های برخورد استفاده می​کنیم.

function ParticleContactGenerator()

{

}

ParticleContactGenerator.prototype.addContact = function(contact,limit){

}

هر یک از این توابع به نوبه خود از درون World فراخوانی می​شوند و هر تماسی را که یافته می​شود، با فراخوانی addContact به World تحویل داده می​شود.

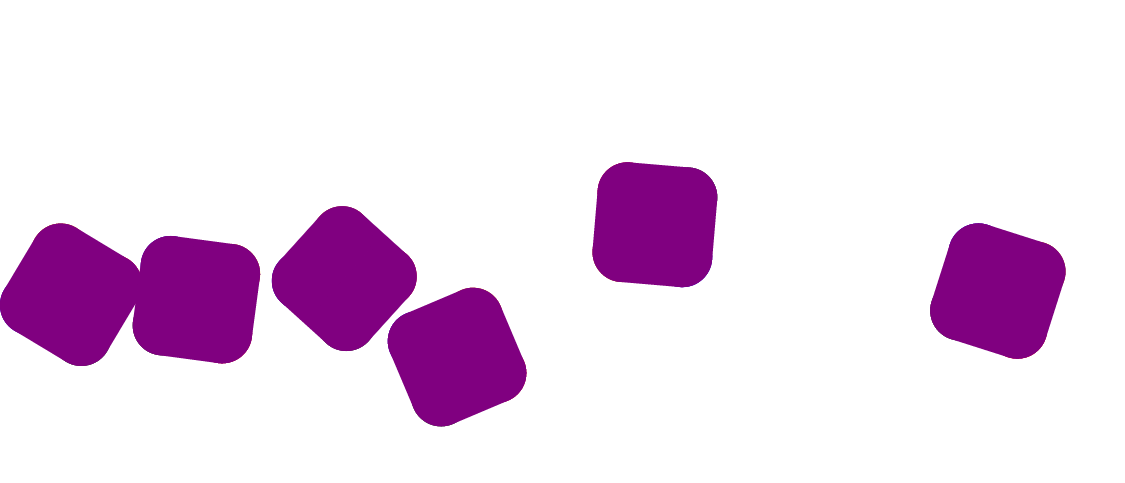
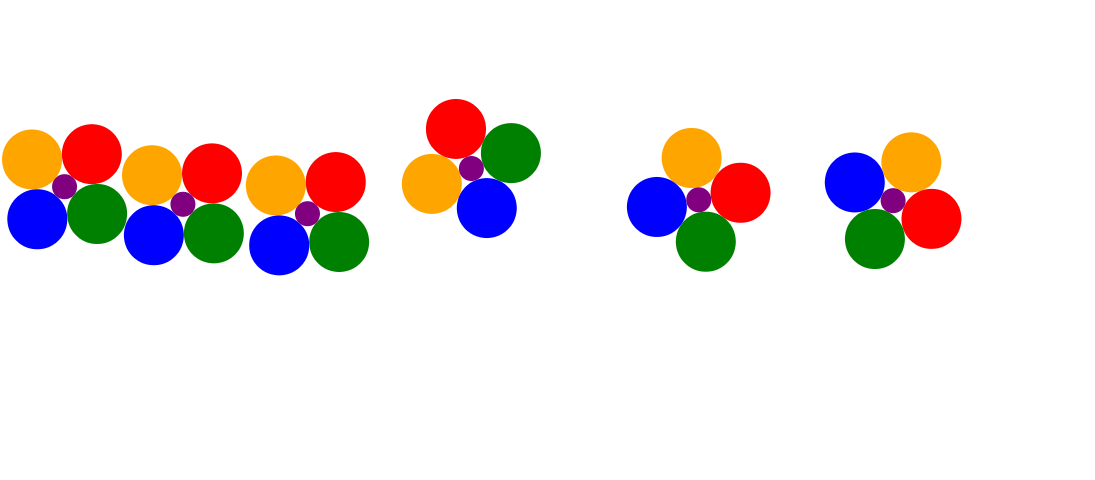
برای اجرای این موتور فیزیک، تابع runPhysics فراخوانی می​شود. این تابع، تمام مولدهای نیرو را با هدف اعمال نیرو فراخوانی می​کند و سپس اجتماع تمام اجسام و یابنده تماس را اجرا می​کند و لیست نتیجه تماس​ها را تحلیل می​کند.

## جمع​بندی و نتیجه​گیری

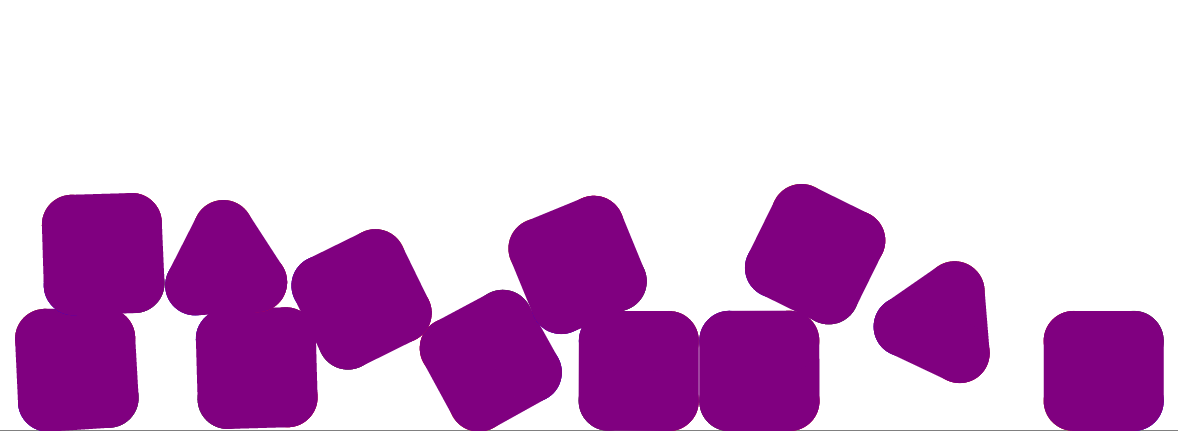
برنامه​های کاربردی زیادی در رابطه با موتور تجمیع اجرام می​توانند ایجاد شوند: ایجاد ساختار​هایی از اجرام ذره و محدودیت​های قوی. با استفاده از این تکنیک، می​توان اجسام بزرگ‌تر بسیاری را ایجاد و شبیه‌سازی کرد. برای نمونه، جعبه​ها، دستگاه​های مکانیکی، و حتی زنجیر​ها و ماشین آلات. اگر فنر​ها را هم در نظر بگیریم، حباب​های نرم و دگر دیس پذیر را هم می​توان اضافه کرد.

در نتیجه یک موتور فیزیک تجمیع اجرام، توانایی شبیه‌سازی چند اثر جالب و پیچیده را دارد. به ویژه، مجموعه​​ای از اجسام نسبتا ساده که توسط ترکیبی از محدودیت​های قابل ارتجاع و سخت، به هم متصل شده اند، نمونه خوبی از استفاده از این رویکرد هستند.

در تصاویر زیر مثال​هایی مهم از قابلیت​های این موتور در ایجاد و مدیریت اجسامی که از تجمیع ذرات ایجاد شده اند آورده شده است توجه شود که برای اتصال ذرات از میله​ها استفاده شده است تا بتوان از چند ذره یک جسم شبیه صلب داشت که حتی بتواند علارقم عدم وجود مکانیزیم درونی چرخش در موتور بچرخد. دو نوع شکل مربع و مثلث تشکیل شده از دایره را در تصاویر زیر مشاهده می کنید:

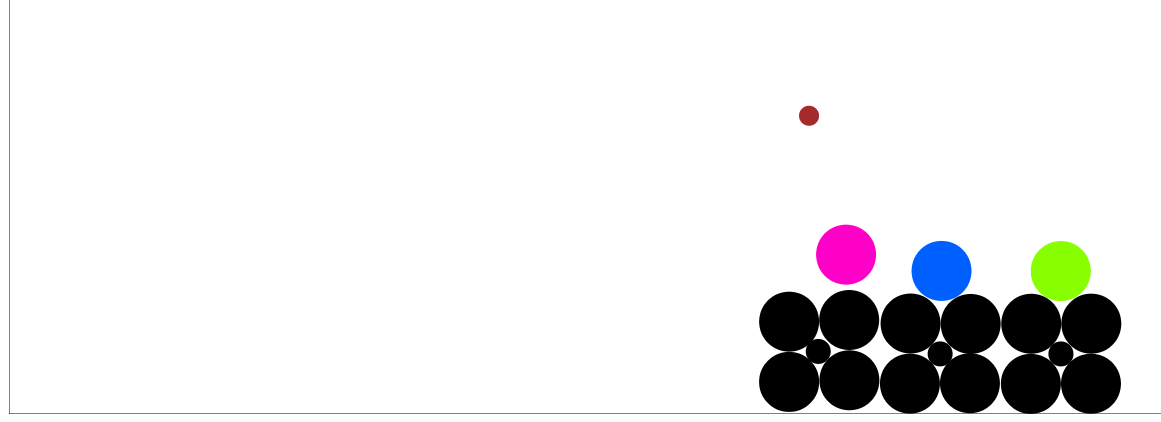
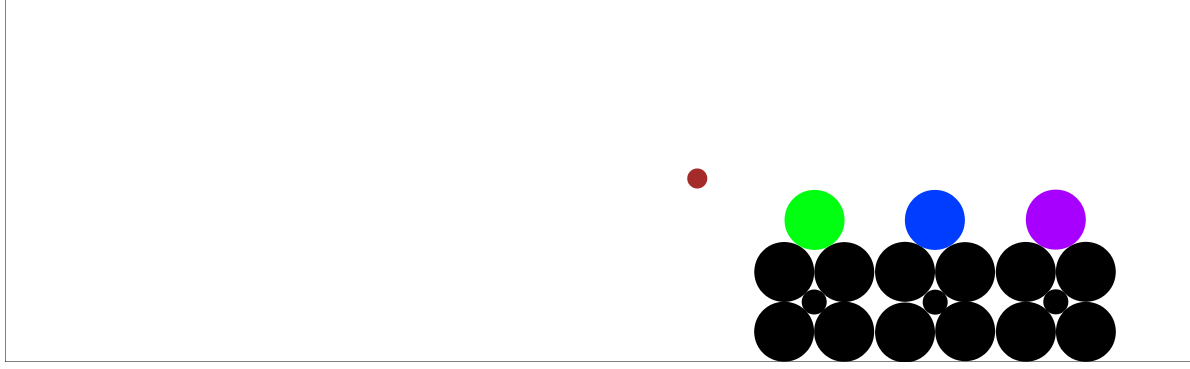


شکل مثال تجمیع اجرام



شکل انواع اجرام تجمیعی

در تصویر زیر همچنان یک بازی ساده ایجاد شده توسط این موتور که در آن بازیکن با پرتاب توپی اهدافی را می زند مشاهده می کنید:



شکل یک بازی با موتور

# نتیجه‌گیری و پیشنهادها

HTML5 به عنوان جدید​ترین نسخه زبان ابرمتنی استاندارد وب در راستای تحقق بیشتر اهداف اولیه اش دارای عناصر بسیاری جدید مهمی​است که انتظار می​رود با فراگیر شدن آن بسیاری از مسائل و مشکلات موجود در نمایش و انتقال اطلاعات در وب که تا کنون فضای اینترنت با آن​ها روبرو بوده است را به سمت راه حل​هایی استاندارد و همه گیر ببرد. یکی از مهمترین این عناصر عنصر canvas می​باشد که در واقع فضایی را در اسناد HTML ایجاد می​کند که بتوان در آن انواع رسوم گرافیکی برداری را داشت. آن چه این عنصر را قدرتمند می​کند قابلیت آن برای پذیرش زمینه​های مختلف است. دو زمینه اصلی موجود برای canvas، 2D Context و webgl می​باشند که هر کدام رابط کاربری خود را برای کاربر ارائه می​دهند. این رابط​های کاربری به کاربر اجازه می​دهند که از محیط canvas استفاده کنند و در آن به رسم تصاویر برداری بپردازند. در این پروژه تمرکز اصلی بر روی زمینه 2D بوده و به توابع و امکانات آن توجه شده است.

مهمترین نکته در کار با زمینه​ها ابراز رابط کاربری آن​ها در زبان JavaScript است، این رابط کاربری با کمک آنچه مرورگر با اسکریپت نویسی به برنامه نویس اجازه می​دهد، این امکان را به وجود می​آورد تا بتوان در محیط Canvas انیمیشن و بیشتر از آن بازی​های ویدئویی ساخت و اجرا کرد. این موضوع سبب شد تا بسیاری از برنامه نویسان تحت وب روی به استفاده از این عنصر و امکانات برای ساخت و ایجاد بازی​های ویدئویی بیاورند. اما به علت نو بودن تمام این تکنولوژی می​توان گفت که هنوز ابزارهای مناسب و با کیفیتی که این پروسه را سرعت ببخشند ایجاد نشده‌اند. یکی از مهمترین این ابزارها در طراحی و ایجاد بازی​های ویدئویی موتور فیزیک می​باشد.

موتور فیزیک در واقع قطعه کدی است که وظیفه شبیه‌سازی پدیده​های فیزیکی در عالم واقع را در بازی​های ویدئویی دارد. بازی​های ویدئویی از این موتور استفاده می​کنند تا آن چه نشان می​دهند برای بازیکن طبیعی‌تر و جذاب‌تر باشد. بازی​هایی که نیاز دارند تا آنچه در دنیا رخ می​دهد را به بازیکن نشان دهند باید از نحوه کار فیزیک اطلاع داشته و ریاضیات آن را در کد خود پیاده​سازی کرده باشند. این ریاضیات می​توانند به طور کلی در کنار بازی قرار بگیرند و نحوه به روز رسانی هر یکی از اشیا در بازی را مشخص کنند. این ویژگی سبب می​شود که بتوان این قطعه کد را از بازی جدا کرد و به طور جداگانه به گسترش و بالا بردن کیفیت آن پرداخت، در نتیجه نه یک بازی که هر بازی که بخواهد از این قابلیت​ها استفاده کند می​تواند از یک موتور جداگانه استفاده کند.

در این پروژه یک موتور فیزیک با قابلیت​های متنوعی برای استفاده در بازی​هایی که در محیط canvas ساخته می​شوند، پیاده​سازی کردیم. برای پیاده​سازی این موتور از فیزیک ذرات شروع کرده و با شبیه‌سازی حرکت آن​ها یک موتور فیزیک ذرات ساختیم. سپس با اضافه کردن مولدهای نیرو و تاثیر تجمیعی آن​ها انواع پدیده​های نیرو زا و وارد کننده نیرو را به موتور اضافه کرده و تاثیر آن​ها را بر ذرات پیاده​سازی کردیم. آن چه که در مورد نیرو و به خصوص نیروهای فنری بحث کردیم روشن کرد که این نیرو​ها در تعداد بسیار زیادی از پدیده​های طبیعی وجود دارند و در شبیه‌سازی آن​ها می​توانند به کار بیایند. اما آنچه بعدا با بررسی قدرت پردازشی کامپیوترها روشن شد این بود که در شبیه‌سازی محدودیت​های قوی با آنکه این محدودیت​ها بسیار شبیه فنرها عمل می​کنند، نمی​توان از این خاصیت استفاده کرد چرا که به خطاهایی بزرگ منجر می​شود در نتیجه یک سیستم مدیریت برخورد به وجود آوردیم که مستقل از مول نیروها عمل می​کند و برخوردها را شناسایی و سپس تحلیل می​کند. این سیستم به ما کمک کرد تا بتوانیم انواع برخورها و پدیده​های مثل برخورد مانند طناب​ها و میله​ها را شبیه‌سازی کنیم.

موتور فیزیکی که در نتیجه در این پروژه پیاده​سازی شد یک موتور فیزیک تجمیع اجرام بوده که می​تواند به عنوان یک موتور فیزیک کامل در بازی​ها مورد استفاده قرار بگیرد اما این موتور همچنان می​تواند پدیده​های بیشتری در طبیعت را در بر بگیرد و حتی تبدیل به یک موتور اجسام صلب شود که در برخی بازی​ها قدرتمندتر عمل خواهد کرد. موتور اجسام صلب برای شبیه‌سازی اجسام بزرگ بسیار کاراتر خواهد بود و در آن به جای این که مانند این موتور اجسام را از ذرات بسازیم می​توانیم آن​ها را به طور یگانه و کلی در نظر بگیریم. برای این کار تغییراتی در موتور باید صورت بگیرد. مهمترین تغییرات نحوه چرخش اجسام است، این موضوع بابت خود باعث پیچیدگی​های زیادی خواهد شد که باعث می​شود از مجال این پروژه بیرون باشد اما در آینده می​توان آن را پیاده​سازی کرد و به این وسیله این موتور را قدرتمندتر ارائه داد.

# منابع و مآخذ

# 

|  |  |
| --- | --- |
| [1] | Y. Cao, "Phusis studio: A real-time physics engine for solid and fluid simulation," *Computational Problem-Solving (ICCP), 2011 International Conference on,* pp. 462 - 465, 2011. |
| [2] | D. H. Eberly, Game Physics, Second Edition, Morgan Kaufmann, 2010. |
| [3] | C. Ericson, Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology), Morgan Kaufmann, 2005. |
| [4] | B. Frain, Responsive Web Design with HTML5 and CSS3, Packt Publishing, 2012. |
| [5] | S. Fulton, HTML5 Canvas, O'Reilly Media, 2011. |
| [6] | D. Geary, Core HTML5 Canvas: Graphics, Animation, and Game Development, Prentice Hall, 2012. |
| [7] | R. Hawkes, Foundation HTML5 Canvas: For Games and Entertainment, friendsofED, 2011. |
| [8] | G. Palmer, Physics for Game Programmers, Apress, 2005. |
| [9] | M. Pilgrim, HTML5: Up and Running, O'Reilly Media, 2010. |
| [10] | I. Millington, Game Physics Engine Development (Morgan Kaufmann Series in Interactive 3D Technology), Morgan Kaufmann, 2010. |

1. impulse [↑](#footnote-ref-1)
2. D’ALEMBERT [↑](#footnote-ref-2)