# 3D 開発や WebGL 開発の基本

## 何事もまずは基本が大切

# 自己紹介

杉本 雅広
(すぎもと まさひろ)

@h_doxas - Twitter

# WebGL スクール概要

当スクールは文字通り、「WebGL に特化したスクール」です。

WebGL の API そのものだけでなく、それに関連した技術についても幅広く扱います。

WebGL は JavaScript を利用してプログラミングを行いつつ、ウェブブラウザでその結果を簡単に確認することができることから、3D プログラミングの学習環境としては他の言語やプラットフォームと比較するとかなり敷居が低いです。

3D プログラミングについて学びたいのであれば、現時点では WebGL は最も優れた選択肢の 1 つだと思います。

とは言え、スクールを進めていく中でみなさんが自然と感じることだと思いますが、3D プログラミングって実は「やらないといけないこと」や「勉強しなければならないこと」がとても広範囲に及びます。

コンピュータグラフィックスの原理や数学・線型代数学など、普通のウェブ開発では出てこない概念も多いです。

ですから、他のプログラミングに関連する技術の場合もそうだと思いますが とにかくトライ・アンド・エラーのサイクルを多くこなす ことがとても大切です。

JavaScript とウェブブラウザで動作確認ができるというメリットを活かして、たくさん挑戦し、どんどん経験を蓄積し、少しずつ手に馴染ませていくのがよいでしょう。

3D プログラミングは（残念ながら）難しいジャンルです。

しかし、ポイントを押さえながら少しずつ基本を埋めていけば、一部の天才にしか扱えないほど特別なものではありません。実際のところ、私自身、学生時代に数学やコンピューターサイエンスを専攻したわけではありませんし、いまだにわからないこともたくさんあります。

それでも、諦めずに続けていけば必ず成果はついてきます。

だからこそ「楽しみながら取り組める状態」を維持するということを、私自身はかなり重視しています。もし「まったく理解できない、つらい……」っていう状況に陥ってしまったとしても、わかることから、少しずつでいいと思います。自分が楽しいと思えること（思える範囲）を重点的に、とにかく 継続 していくことが肝要です。

# 基本的な運営方針

- 講義中に Twitter 等の SNS に発言しても OK
- 講義後にブログや Zenn, Qiita, note などでアウトプットしても OK
- スクールのサンプルからコード流用しても OK（ライセンスフリー）
- サンプルファイルだけを見れば内容がわかるようにコメントを記載してあります
- 概念や考え方はスライドで、実装はサンプルで詳しく解説

> スクールの期間が終わっても質問は期限なくずーっと受け付けます ✨🚀✨

# WebGL とはなにか

さて、それでは早速ですが WebGL とはそもそもなんなのか、から考えていきましょう。

実際のところ、WebGL とはなんであるか——そのことを厳密に理解しておかなければならないってことは別にないのですが…… とはいえ、やはり最初ですのでそこから話をしていきます。これについては簡単に概要だけ理解していれば十分ですので、まずはざっくりと WebGL が何者なのか理解しておきましょう。

WebGL は、3DCG を描画するためのグラフィックス API として広く利用されている OpenGL のファミリーに属しています。

OpenGL、みなさんは聞いたことがあるでしょうか。これは、PC だけでなくスマートフォンやタブレットなどにも搭載されている「グラフィックスを描画するための API」で Khronos という非営利団体によって管理されています。


> OpenGL などを管理する Khronos が WebGL の仕様を管理している

現代では PC はもちろん、スマートフォンなどでも 3DCG で描かれるゲームを遊ぶことができますよね。

これは OpenGL などのグラフィックス API と、それを解釈するドライバが各種端末にあらかじめインストールされているからこそ可能なことです。世界標準である OpenGL（とそのシリーズ）が存在するからこそ、ソフトウェア開発者はある程度統一した仕様に沿ってグラフィックスを描画するプログラムを開発できるわけですね。

> 家庭用ゲーム機も、中身は OpenGL というのは普通にあります

WebGL は、そんな OpenGL シリーズのなかの 1 つであり、ウェブにおける CG 描画の標準機能として利用できます。

より具体的には、WebGL は OpenGL のシリーズのうち「OpenGL ES 2.0」という API をベースにした仕様となっていて、使い方もそれと非常に似ています。メソッド名や手続きなどはほとんど同じ、と言ってもいいでしょう。ちなみに OpenGL ES 2.0 はほとんど全てのスマートフォン、タブレットなどが対応している「モバイル向けの軽量な OpenGL」の実装です。


> WebGL はモバイル向けの軽量な実装である OpenGL ES 2.0 相当の機能を持つ JavaScript の API

さて、OpenGL や、OpenGL ES は、たいていの場合 C++ や Java など、それぞれのプラットフォームに合わせてネイティブな言語で記述されます。

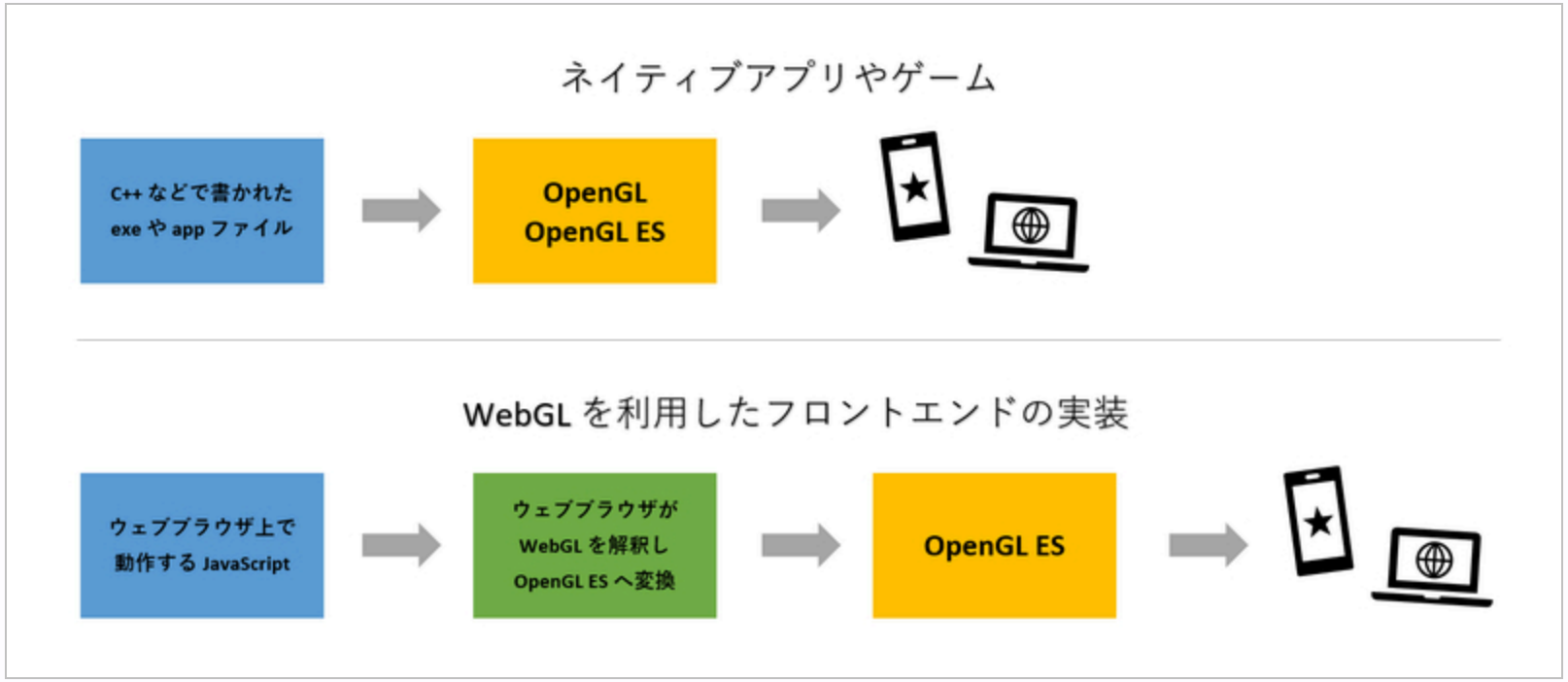
WebGL はと言うと、当然ながら JavaScript で記述することになります。ウェブ関係の開発を行っている人が普段から使い慣れている JavaScript を使って、ブラウザ上で OpenGL の API を叩く（に相当する）ことができるというのが WebGL が持つ優位性だと言えます。

> ただし OpenGL ES 2.0 自体は 2007 年公開なのでかなり古い API

JavaScript で記述された WebGL のコードは、最終的には（その JavaScript を解釈しているウェブブラウザによって） OpenGL の API として実行されます。

これにより、通常の OpenGL と同様に GPU を利用したレンダリングが行われるため、非常に高速な描画が実現できます。GPU は、Graphics Processing Unit の略称で、グラフィックスの描画に特化したハードウェアです。

> 実は Windows の場合はちょっと違います（後述）




GPU を使うなど原理そのものが違うので高速に動作するのであって、WebGL を使ったからといって JavaScript が高速化するわけではない点に注意

# つまり WebGL とは

- WebGL とは OpenGL ES を JavaScript から叩ける API 群のこと
- アプリケーション（JS）は CPU 上で動作するが GPU の動作は OpenGL ES と同等
- 言い換えると OpenGL ES をそのまま記述するのと基本的に同じことを行う必要がある
- 従来のウェブの常識が通用せずネイティブアプリのようなコードの記述スタイルになる
- 逆に言えば、WebGL を使いこなせるなら他のプラットフォームで OpenGL を使うのは難しくなくなる

# 余談コラム

Windows 環境では、Google Chrome に組み込まれている ANGLE が、WebGL のコードを DirectX 用に置き換えて実行するという特殊な現象が起こります。

ANGLE は Google が開発を手がけるプロジェクトで、Google Chrome で「各ハードウェアに搭載された OpenGL などのドライバに依存せずに確実に WebGL を実行するため」に、DirectX で WebGL を動かしてしまえばいいじゃない！ というかなり豪腕な感じのプロジェクトです。

google/angle - GitHub

# three.js と WebGL

さて、それではみなさんも恐らくご存知の、WebGL ラッパーライブラリである three.js と WebGL との関係はどのようになっているのかについても見てみましょう。

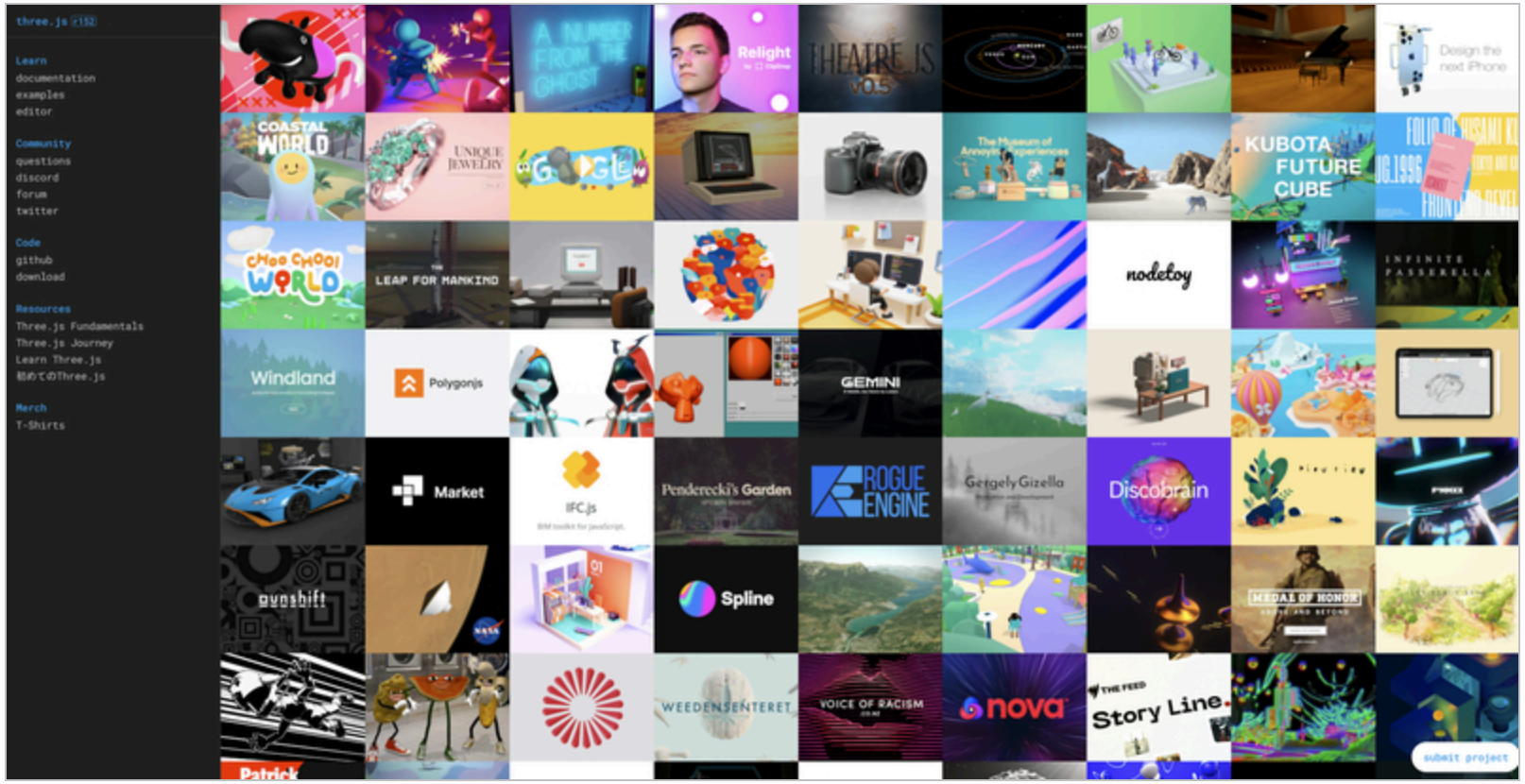
既にそのあたりはご存知の方も多いかもしれませんが、最初なのでちょっと丁寧にいきます。

three.js は Mr.doob によって開発された WebGL コンテンツの実装を助けるライブラリです。

正確には、様々な機能をもっているため必ずしも WebGL 専用のライブラリというわけではありませんが、現在はほぼ WebGL 実装には欠かせない 3D ウェブの jQuery 的存在となっています。

three.js を利用すると、私個人の体感的には、難易度的にも実装工数的にもピュアな WebGL を 100 としたら 10 〜 20 ぐらいまで開発者の負担を軽減できる印象です。

また、豊富なサンプルやドキュメントも用意されています。それらのことを踏まえれば、普通に考えると three.js を使うのが今現在では最も手軽な WebGL の利用方法だと言えると思います。




# Three.js – JavaScript 3D Library

# three.js のメリット

- 手軽に、かつ効率的に WebGL を記述できる
- 多彩で強力な機能を持ち拡張もできる
- 情報が（比較的）豊富にある
- 日本語で解説されていることも多い（が、バージョンの違いには注意）
- 現在も活発に開発が続いている

# three.js のデメリット

さて、three.js が極めて便利なライブラリであることはわかりましたが、一方で three.js を使うことによって生じるデメリットはあるんでしょうか。

これについては WebGL とどう向き合っていくのかによって、考え方は様々かと思いますが…… その享受できるメリットの大きさに比べれば、デメリットはほとんどありません。

あえて無理やりデメリットをひねり出すと……

- three.js の流儀に従う必要がある
- ややファイル容量が大きめ
- バージョンアップに追従するのが大変（破壊的変更も割りと普通にあります）
- 成果物の見た目が似通ってくる

たまに勘違いされてしまうことがあるのですが、私自身は three.js の否定派というわけではありません。

むしろ、これほど手軽に WebGL を扱うことのできる three.js という存在がなければ、今ほど WebGL が認知されることもなかったと思っています。ですから、みなさんが three.js を使うことに後ろめたさとか、未熟さとか、そういうものを抱く必要はありません。

ただし、これは最初にしっかり理解しておいてほしいのですが、WebGL はその API が難しいというだけではなく 純粋に 3D プログラミングそのものが難しいジャンルである というのが見方としては正しいです。

three.js はそういった 3D に独特な難しさを良く言えば隠蔽してくれていますが、同時に（ちょっとうがった見方をすると）本質を理解することを阻害しているというふうにも言えます。要は three.js を使っていると 3D の基礎を知らなくても割となんとかなってしまう のですよね。

この WebGL スクールでは、three.js が良くも悪くもうまく隠蔽してくれている部分を、ネイティブな WebGL の話も踏まえながら明らかにしていくスタンスで進めていきます。

まずは three.js を使って「動く状態に手早く持っていき」つつ、「3D プログラミングそのものに慣れる」ようにしましょう。そしてスクール後半ではネイティブな WebGL の実装も併せて扱いますので「本質的な理解を深め汎用性と応用力の高い基礎知識を蓄積させていく」ことが理想です。

# WebGL スクールを通じて学んでほしいこと

- three.js なら手軽に動くところまで持っていきやすい
- まずは「とりあえず動いている」という事実はとても重要
- 動いている実装が目の前にありさえすれば比較や検証が行える
- ただし three.js だけではどうしてうまくいかないのかを本質的には理解しにくい
- そこを掘り下げネイティブな WebGL の実装で基礎 3D 力を養うことこそが本スクールの最終目標

# three.js 実習

本講義のカリキュラムでは、最初のうちは three.js だけを使います。

これは先述のとおり まずは 3D に慣れるということが非常に重要 だと私が考えているからです。後半になると逆にネイティブな WebGL の API を利用したサンプルを使っていく感じになりますが、まず最初に 3D の開発とはどういうものかということを three.js でつかんでおくことはとても大きなプラスになるはずです。

何度も同じことを言うようですが、大事なのは **自身の手で動かすこと** や **楽しいと思える範囲で継続して** 取り組んでいくことです。

サンプルはそれなりに数が多いですが、それにみなさん自身が手を加え、動かし、検証していかなければ、難易度だけが一方的に上がっていってしまいます。コメントを自分の言葉で書き添えるだけでも、最終的な理解度は変わってきます。どんどん積極的にサンプルを触ってみてください。

> もちろん質問はいつでも歓迎です！

# サンプルの解説やコメントについて

スクールのサンプルでは、コード内のコメントを重視した作りになっています。

ただ、これは進めていくうちに自ずと明らかになってくるのですが、WebGL や 3D プログラミングはコードが非常に長くなりがちな面があるので、過去に解説した部分についてはコメントを消して、そのサンプルにおける重要なポイントにフォーカスしやすいようにしてあります。

１番を改造して２番、２番をさらに改造して３番、というように少しずつ修正しながら内容が充実していくような過程においては、前回のサンプルとの 相違点の部分 や 注目すべき部分 に、アットマークを使って目印をつけてあります。

コメントのなかに `@@@` というように３連アットマークが出てきたときは、そこが以前のサンプルから修正された場所、あるいはそのサンプルにおける重要なポイントを示していると思ってもらえるといいと思います。

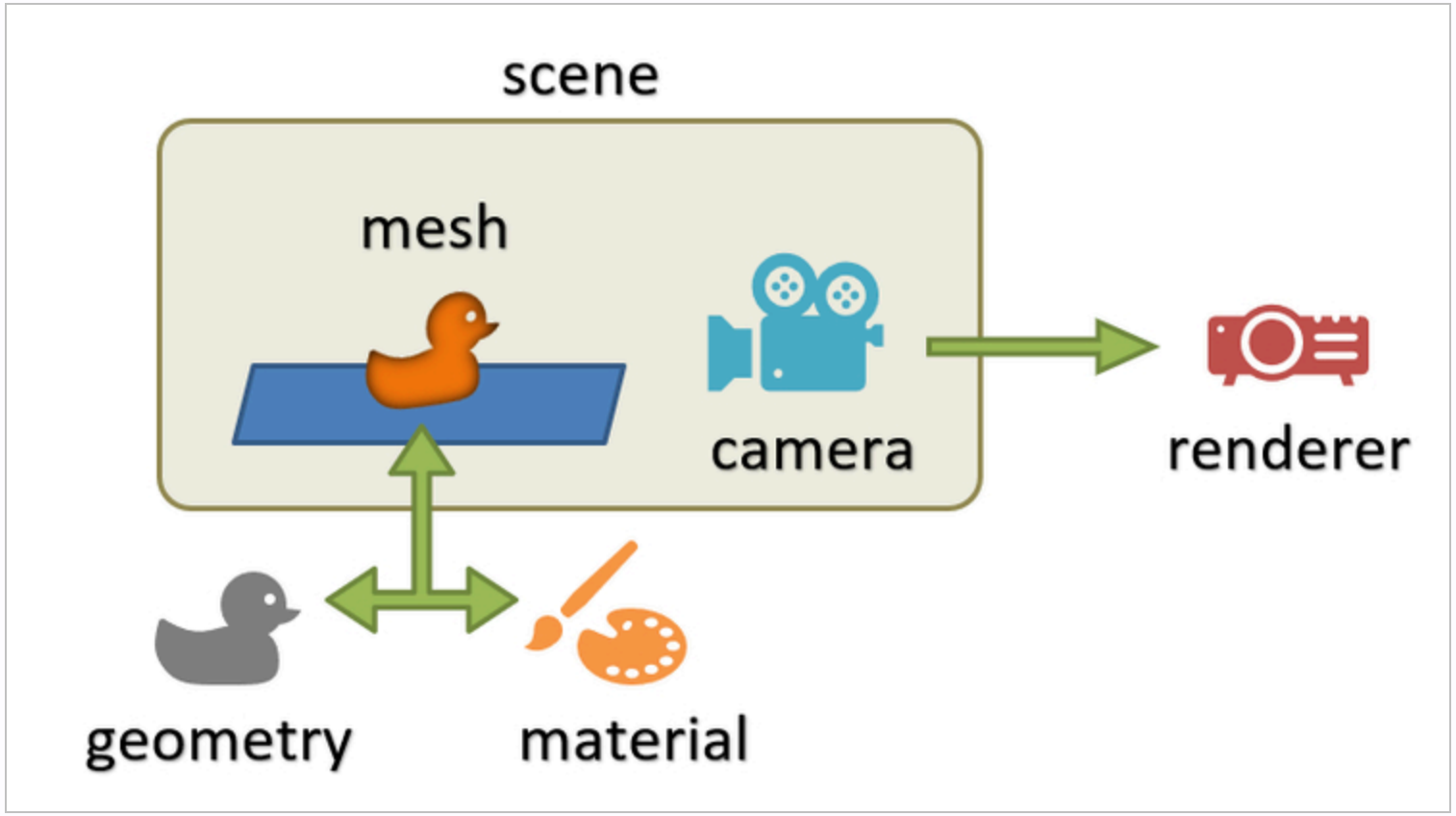
また、今見ているようなスライドでは、主に概念や図式で理解したいポイントを重点にしています。

両方を同時に見ながらというよりは、まずスライドで概要を掴んだ上で、サンプルのコードを読み解いていくのがいいでしょう。コメントは自分なりの言葉で積極的に書き足したり修正したりしながら、見返したときにポイントを掴みやすいようにしておきましょう。

# 001

- モジュール形式で記述する
- 可能な限り変数の宣言には const を使う（再代入不可）
- 大文字のみで構成される変数・プロパティは定数的に利用する
- three.js が持つクラスをひとつの指標として役割を覚えていこう
- three.js では色の表現を 16 進数で表記・指定できます（色についての参考： Color – three.js docs）

> three.js については公式ドキュメントや、場合により最新のコードを確認するのが鉄板です




ここではまず、それぞれの言葉の意味をイメージできるようにしよう

# 002

- マウスで操作しながら確認できるように準備する
- コントロールにはカメラとイベントを拾う対象となる DOM を渡す
- 恒常ループを記述してアニメーション処理させる
- アニメーション処理には `requestAnimationFrame` を用いる
- JavaScript 特有の `this` の挙動に気をつける

# 余談コラム

`requestAnimationFrame` は「ブラウザが画面のリフレッシュレートと自動的に同期を取ってくれる」という特徴を持った「アニメーション処理を実現するためのメソッド」です。

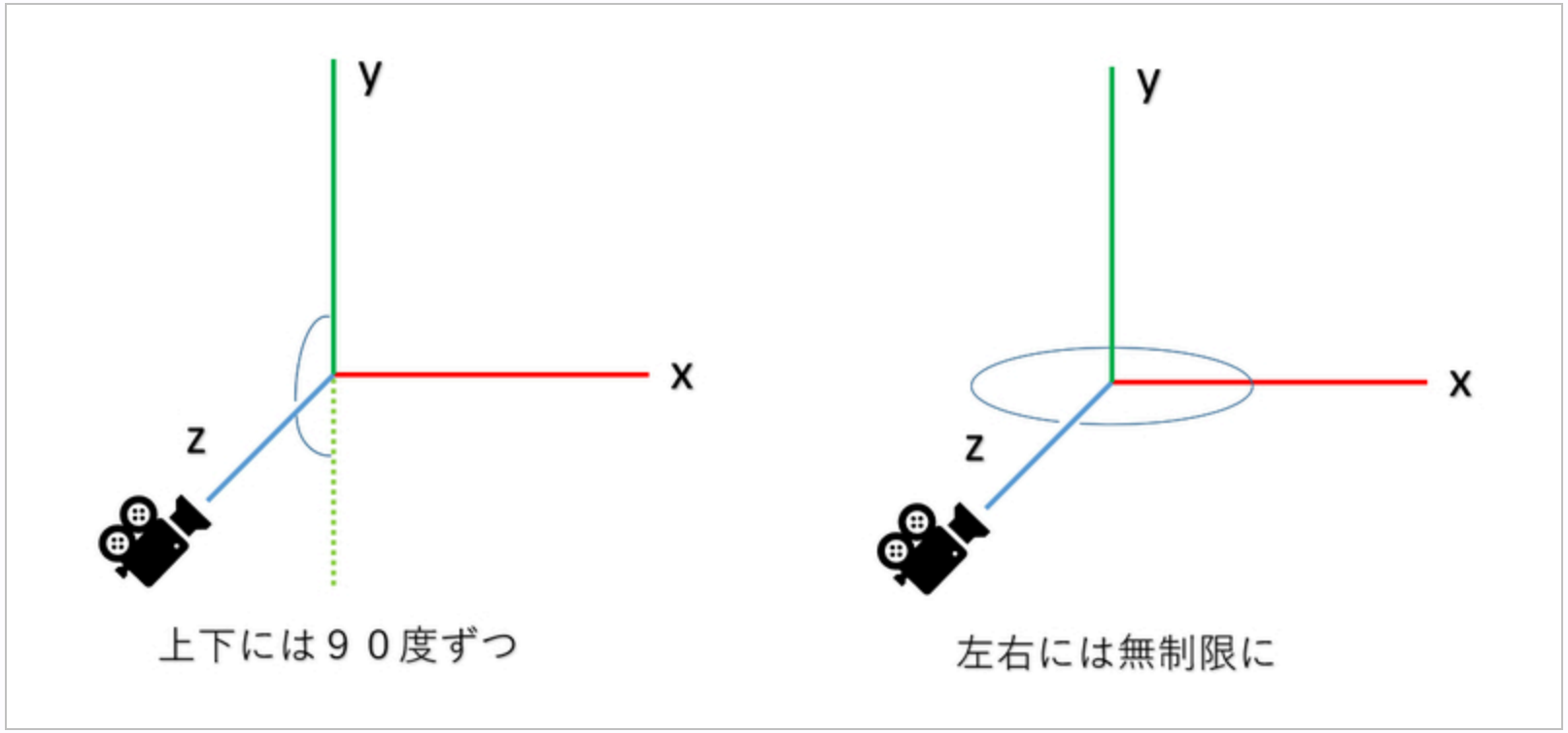
通常、一般的なディスプレイは約 60fps で画面を更新しています。つまり、1 秒間に 60 回画面を更新するわけです。`requestAnimationFrame` を利用すると、この「ディスプレイの更新間隔」に合わせて同期を取りながら関数呼び出しが自動的に行われるようになり、結果なめらかに動くアニメーションが実現できます。

> 負荷が高すぎる場合はそのとおりにならないことも

また、three.js のオービットコントロールは、手軽にインタラクティブに操作できるカメラを実現できる優れた手段です。

一般的にオービットコントロールがやってくれているような「マウスで操作できるカメラ」を実現するのはなかなか難しいです。このあたり、もし腕に自信があるのであれば自力で実装してみてもいいかもしれません。

> three.js には多くの算術関数があるので意外といける……かも？




# OrbitControls の可動域のイメージ

# 003

- three.js にはヘルパーと呼ばれる補助機能がある
- ヘルパーは、開発に便利な目印やラインなどの 3D オブジェクト
- 開発中はヘルパーを導入しておくとなにかと理解もしやすい
- 他の 3D オブジェクトと同様、シーンに対して `add` することで追加できる

> XYZ の軸が表示されているだけでも立体的な空間を把握しやすくなります

# 004

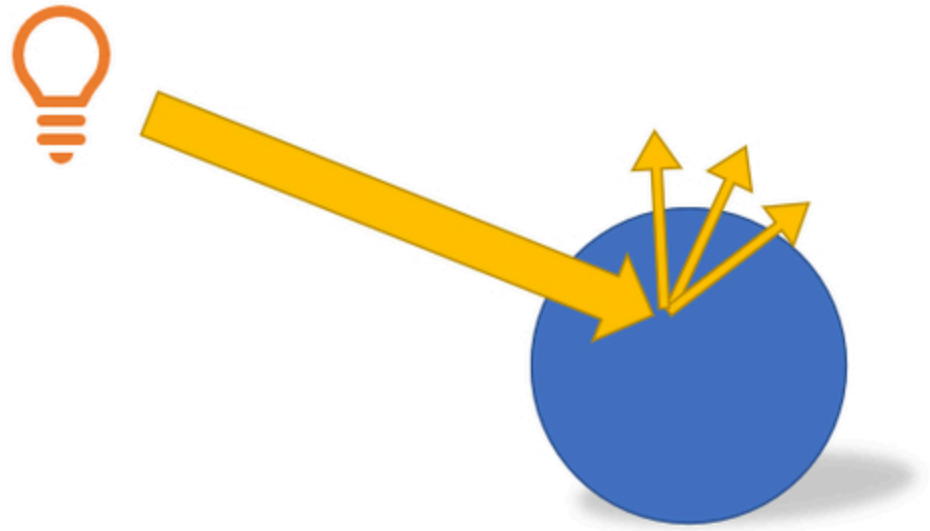
- JavaScript のイベント処理を 3D シーンに反映させる
- キーの押下を検出する `keydown` などをうまく活用
- three.js では `Object3D` という基底クラスがある
- このクラスに属するインスタンスは皆 `rotation` などの便利なプロパティを持つ
- メッシュやカメラは、いずれも `Object3D` を継承している

参考：Object3D – three.js docs

# 005

- 3D における「ライト」には様々な種類がある
- three.js の場合もそれは同様で、ライトには複数の種類がある
- ここでは平行光源（ディレクショナルライト）を使う
- three.js ではライトを追加するだけでなく、その影響を受ける側（マテリアル）の設定も必要な点に注意する
- ディレクショナルライトの位置設定は、最初はイメージするのが難しいが「指定した座標のほうから原点に向かって光が降り注いでくる」というふうにイメージすると良い

平行光源によるランバートライティングは 拡散光 とも言います。



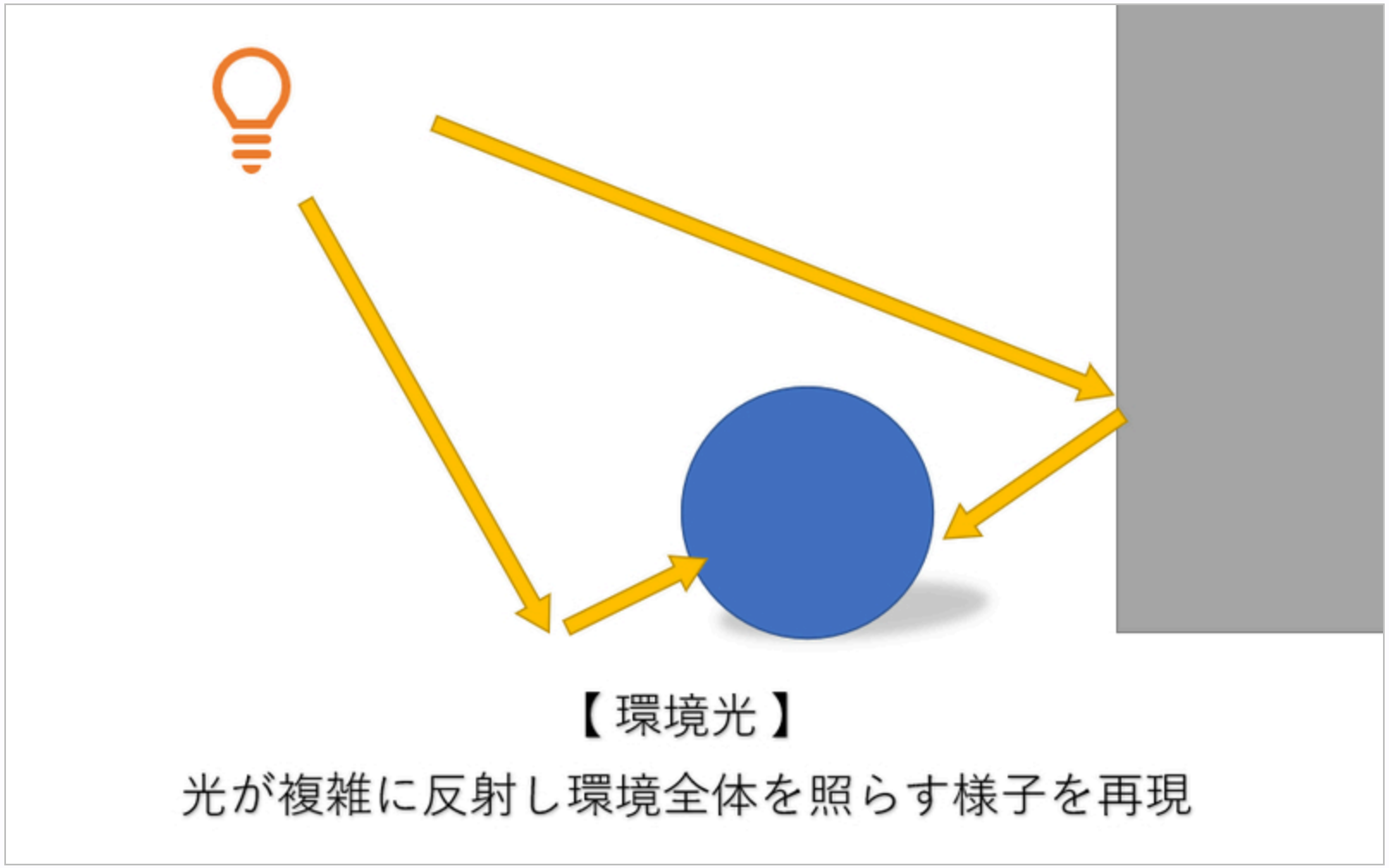
【 拡散光 】

光が衝突し、拡散する様子を再現

# 006

- アンビエントライト（環境光）をシーンに新たに追加
- 環境光は現実世界に近づけようとすればするほど高難易度・高負荷になる（フォトリアルなレイトレーシングなど）
- リアルタイム 3D の場合は（負荷を抑える必要から）色を加算することで近似することが多い
- Phong のシェーディングモデルを扱うマテリアルに変更することで、反射光の効果が表れるようになる
- 3DCG 的には、拡散光と反射光は似ているが視線を考慮するかどうかなど違いがある

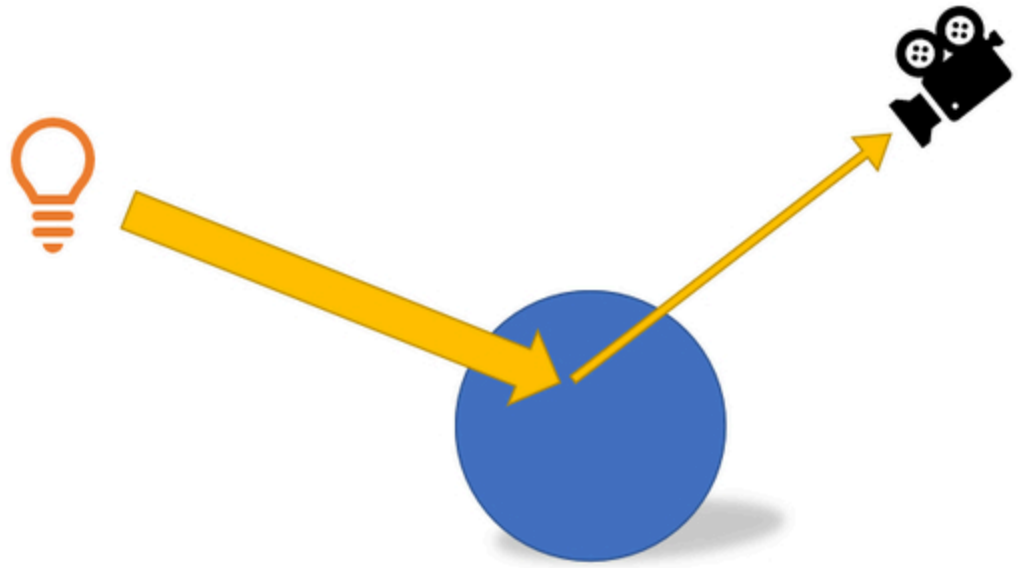
アンビエントライト（環境光）はイメージ図にするとこんな感じ。



環境光は、現実世界ではあまりにも当然のように存在しますが CG で再現するのがかなり難しいもののひとつです。現実世界では、いわゆる間接照明みたいに、直接光源が目に見えない位置にあっても光は複雑に反射しながら漏れ出してきます。これを CG で再現するには、真面目に全部を計算するか、偽物の方法で似せるか、どちらかしかありません。

一般に、真面目に全部を計算するのはかなり負荷が高いので、リアルタイム CG では色を加算したりして近似することが多く、これをアンビエントライトと呼んでいます。

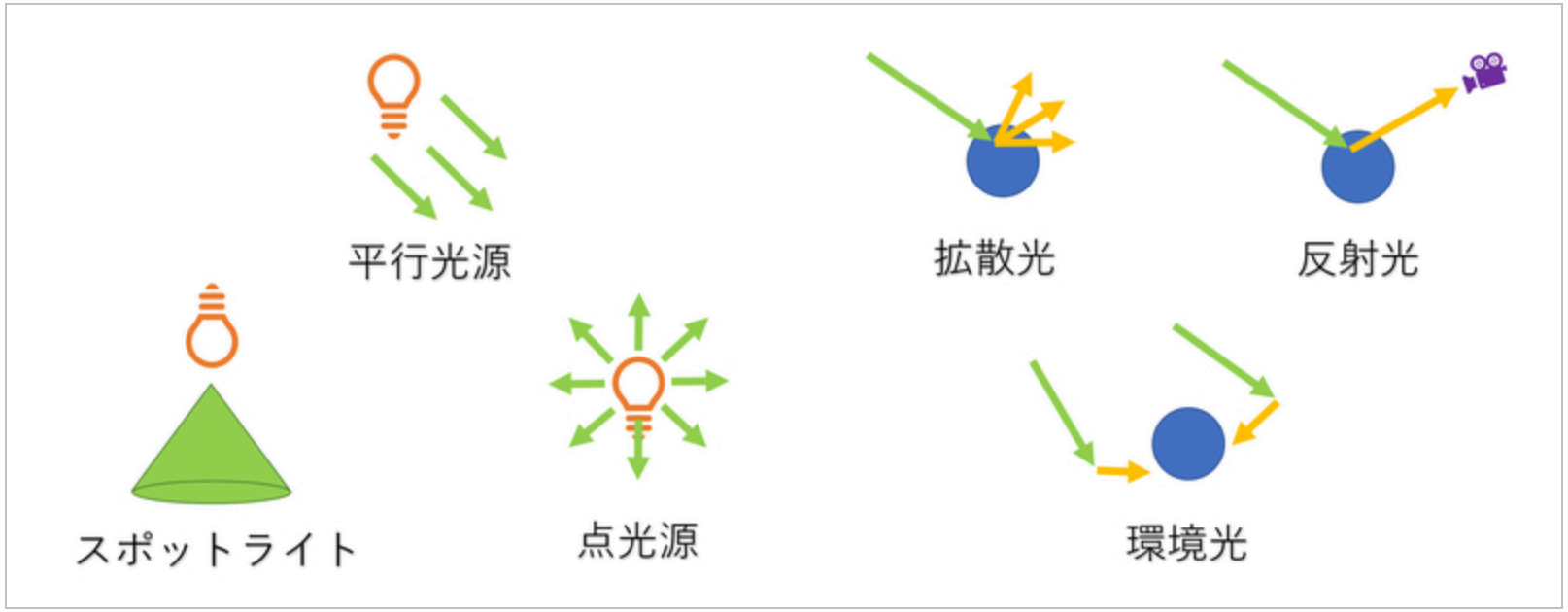
反射光のイメージ図はこんな感じ。



【反射光】

光が反射し、視点に届くかどうかを考慮したハイライト

光源の種類が複数あること、そして拡散光や反射光など複数の照明効果があることをイメージできるようにしておこう。



# 007

- three.js には様々なジオメトリがあらかじめ用意されている
- ジオメトリはいわば「骨格」あるいは「形状」、「設計図」と考えるとわかりやすい
- 異なるジオメトリからメッシュを作ると当然描画結果も変わる
- ジオメトリ生成は形状によってパラメータが異なる
- 意味がわからない場合は公式のドキュメントや Example を見よう

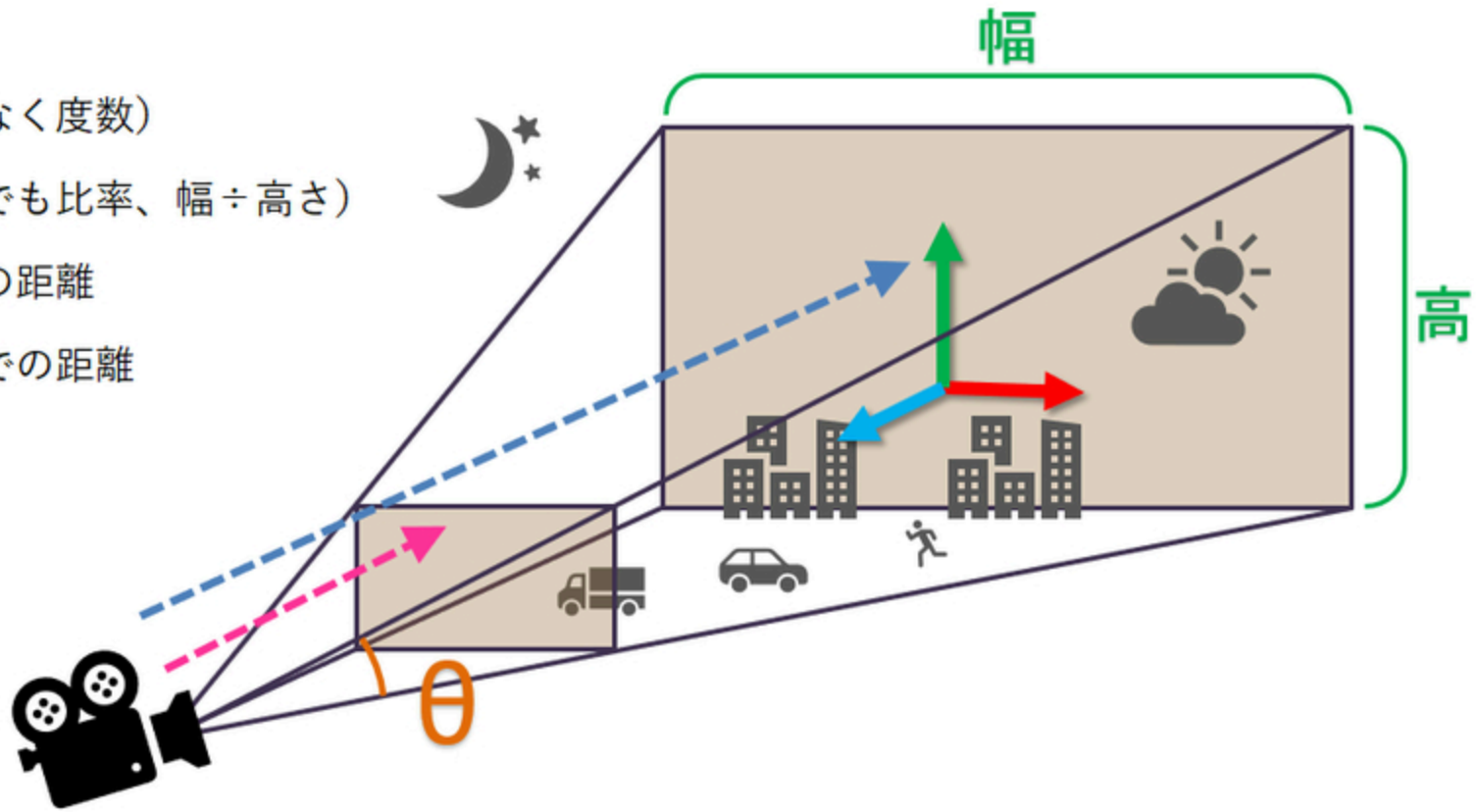
# 008

- カメラ関連のパラメータの意味を把握しておく
- 丸暗記することにはあまり意味はないがキーワードの意味がわかっていることの意味は大きいので、できればある程度は覚えたい
- ウィンドウ全体にフルスクリーンで描画したり、ウィンドウサイズが変更することに対応したりする上で、いくつもサイズに関する設定が必要なので要注意！

> カメラ関連のパラメータは次ページの図も参考に

| | |
|---|---|
| fovy | θの部分の角度（ラジアンではなく度数） |
| aspect | クリップ空間の縦横比（あくまでも比率、幅÷高さ） |
| near | カメラからニアクリップ面までの距離 |
| far | カメラからファークリップ面までの距離 |

*ニアクリップ面とファークリップ面の間の、*
*視錐台に囲まれた空間内だけがレンダリング*
*の対象になり、範囲外のものは画面には出ない*
*※つまり月は範囲外なので画面には描かれない*



# 009

- 同じ骨格（ジオメトリ）を使い回せる場合は、使い回す！
- 何度もジオメトリを生成してしまうというやらかしは、特に不慣れなうちはしてしまいやすい
- ジオメトリとマテリアルは使い回せることを覚えておく
- JavaScript では `Math.random` で乱数（ランダムな値）を得られる
- `Math.random` の戻り値は 0 以上 1 未満

# まとめ

さて、第一回はこのあたりまでにしましょう。

今回は WebGL とは何かといった API の背景に始まり、three.js を使ったサンプルを通じて 3D 表現を行うための基本について見てきました。かなり基本的な話ばかりでしたが、それでも結構盛りだくさんだと感じた方も多いかもしれません。

3D プログラミングはとても難易度の高いジャンルではありますが、three.js などのラッパーライブラリを活用すれば、その本質的な難解さを回避しつつ、手軽にコンテンツの制作を行うことができます。

今回利用したサンプルは本当に基本的なものでしたが、これらに登場した概念を元に、いろいろと工夫を重ねるだけでもそれなりに見栄えのするものを作ることができるはずです。

> 自分の手の中にある理解できたものを組み合わせるだけでも作品は作れる！

さて、最後になりますが、WebGL スクールでは毎回（できるだけ）課題を出したいと思っています。

課題の提出は強制しませんが、Discord 上に課題提出用のチャンネルを作ってありますので、そこで完成した課題をシェアしてもらえると嬉しいです。

誰かに作品を見てもらったり、感想を伝えてもらったりするのってすごく大きなモチベーションや刺激になります。

自分で課題の提出をすることが難しくても、せめて他の方々の作品を見かけたときは感想などを送って相互に刺激を受けられる環境にしていけたらいいなと思ってます。

## 今回の課題の要件

- Box Geometry を利用すること
- ボックスが画面上に１００個以上描かれるようにすること
- まっすぐ並べてもいいし……ランダムでもいい（配置の仕方は自由）
- 色や大きさなどは自由

> 上記の要件を満たす技術デモを作ってみよう

最初のほうにも書いたとおり、3D プログラミングに限らず、あらゆる学習は 自身で手を動かしながら、失敗や成功を繰り返していく なかで身についていきます。

作業時間を捻出するのは大変だと思うのですが、簡単なものでも全然構いません。ちょっとだけがんばって、自分なりに実装を作ってみてください。実装していくなかでいろいろ疑問も出てくると思いますが、Discord 等でいつでも質問してもらって大丈夫です！