

# **SceneKit BOOK**

**for Swift Playgrounds**

# **SceneKit Book**

**Toshihiro Goto 著**

**2017-10-22 版 x67x6fx74fx6f 発行**

# はじめに

---

SceneKit というマニアックな題材の本を手に取っていただきありがとうございます。

iOS の 2DCG フレームワークである SpriteKit 関連の本は書店に置かれていますが、3DCG のフレームワークである SceneKit に関してこのフレームワーク中心に描かれている本があまりないので、書いてみようと思いついたものがこちらとなります。

また、動作を行う環境として幅広く使用できる Swift Playgrounds を選びました。Xcode がインストールされていれば Mac 版の Swift Playgrounds でも使用でき、ViewController や View をコピーすると通常のアプリでも使用できるためです。

決して、Xcode の Scene Editor のスクリーンショット画面を撮ることが面倒というわけではありません。決して。

## この本で **Swift** や数学の知識は必要？

書いている本人がアホなので、あまり必要はないです。

## iPad は必須？

iPad の Swift Playgrounds を使用していますが、ほとんどのコードは macOS 上で動く Xcode の Swift Playgrounds でも動作可能です。注意点として Xcode の Swift Playgrounds は iOS Simulator 同様に Metal ではなく OpenGL で操作しており、一部機能が使えません。

Xcode の Playgrounds で Metal を使用する場合、macOS のアプリとして作成する必要がありますが、構文はほぼ変わりません。

## Swift Playgrounds 動作環境

- iOS11 の iPad mini 2、iPad mini 3、iPad mini 4
- iOS11 の iPad Air、iPad Air 2、全ての iPad Pro
- Xcode 9 が動作する全ての macOS High Sierra の Mac

iPad は ARKit の一部機能とジオメトリのテッセレーション関連が A9 以降（2 番目のもの）で動作し、Mac では ARKit が動作せず、Swift Playgrounds の一部機能は macOS High Seirra でのみ動作します。

## この本での検証環境

iPad Pro 12.9 インチ（2016）と iPad Pro 10.5 インチ（2017）  
古い端末では負荷の問題で動作しないコードもあるかもしれません。

## 参照元と本書の内容

過去に書いていた Blog 記事<sup>\*1</sup> を iOS 11 用に直したものと、新機能の紹介となっています。

## 本書のサンプルコードと画像などのリソース

ページの関係上記載されているコードが少ないため、以下の URL からサンプルコードを取得してください。本書で使用する画像などのリソースも含まれています。

[https://github.com/ToshihiroGoto/techbookfest3\\_SceneKit](https://github.com/ToshihiroGoto/techbookfest3_SceneKit)

ダウンロードする場合は「Clone or Download」を押して、「Download ZIP」を押してください。

---

\*1 <http://appleengine.hatenablog.com/archive/category/SceneKit>

## 免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行なってください。これらの情報にやる開発、製作、運用の結果について、著者はいかなる責任も負いません。

# 目次

はじめに	1
この本で Swift や数学の知識は必要？	1
iPad は必須？	1
Swift Playgrounds 動作環境	2
この本での検証環境	2
参照元と本書の内容	2
本書のサンプルコードと画像などのリソース	2
免責事項	3
<b>第 1 章 3DCG とは？</b>	<b>25</b>
1.1 SceneKit を始める前に 3DCG で必要なものとは？	25
1.1.1 カメラ	25
1.1.2 ライト	25
1.1.3 オブジェクト	25
1.1.4 マテリアル	25
1.2 リアルタイムレンダリングとオフラインレンダリング	26
1.3 ワールド座標とローカル座標	26
<b>第 2 章 Swift Playgrounds for iPad の使い方</b>	<b>28</b>
2.1 アプリの入手	28
2.2 空白（Blank）の Playground ページのつくり方	29
2.3 UI	32
2.4 プレイグラウンドページ編集	33

# 目次

---

2.5	リソースとコードスニペット . . . . .	33
2.5.1	コードスニペット . . . . .	33
2.5.2	リソース . . . . .	34
2.6	ツール . . . . .	36
2.6.1	詳細 . . . . .	36
2.7	ステップ実行への変更 . . . . .	38
2.8	コード実行 . . . . .	38
2.9	アンドゥ、リドゥ . . . . .	39
2.10	キーボードを閉じる（外部キーボードのみ） . . . . .	39
2.11	コードエラー . . . . .	39
2.12	ヘルプ . . . . .	40
2.13	ナンバーパット . . . . .	42
2.14	カラーリテラル . . . . .	42
2.15	イメージリテラル . . . . .	43
2.16	Playground ファイルの検索とソート . . . . .	44
<b>第 3 章</b>	<b>iOS アプリの概要</b>	<b>46</b>
3.1	Swift Playgrounds for iPad での iOS アプリの仕組み . . . . .	46
3.1.1	Swift Playgrounds for iPad と Xcode で作成する際の違い . . . . .	46
3.2	試しにアプリをつくる . . . . .	46
3.3	コードを軽く変更する . . . . .	48
3.3.1	変更するコード . . . . .	48
<b>第 4 章</b>	<b>Xcode 版 Playgrounds</b>	<b>50</b>
4.1	Xcode 版の Playgrounds で新規の Playground ファイル . . . . .	50
4.2	Xcode の Playgrounds で使用するリソースの保存方法 . . . . .	52
4.3	iPad で作成した Playground ファイルを Xcode の Playgrounds で実行する . . . . .	53
4.4	Xcode 版の Playgrounds のビルド、実行が遅い場合 . . . . .	54
<b>第 5 章</b>	<b>Xcode で macOS アプリを作成する</b>	<b>55</b>
5.1	iOS から macOS に変更する際に、修正する命令 . . . . .	55
5.2	Project の作成 . . . . .	55

## 目次

---

5.3	コードを書く	57
5.4	実行結果	59
<b>第 6 章</b>	<b>SceneKit の概要</b>	<b>60</b>
6.1	SceneKit とは？	60
6.2	対応プラットフォーム	60
6.3	SceneKit の中身と関連するフレームワーク	60
6.3.1	SceneKit の中身	60
6.3.2	関連するフレームワーク	61
6.4	SceneKit 座標系について	61
6.5	OpenGL (ES) と Metal	62
6.5.1	OpenGL	62
6.5.2	OpenGL ES	62
6.5.3	Metal	62
6.6	Xcode での SceneKit	63
6.6.1	Scene Editor	63
6.6.2	Debug	64
<b>第 7 章</b>	<b>ほんの少しの数学の話と 3DCG のプログラム使用する技術</b>	<b>66</b>
7.1	ラジアン (rad)	66
7.2	行列	66
7.3	法線	67
7.4	UV マップ (Texture Coordinates)	68
<b>第 8 章</b>	<b>SceneKit の要であるシーディングラフ</b>	<b>69</b>
8.1	シーディングラフの構造	69
8.2	シーディングラフでの注意事項	70
<b>第 9 章</b>	<b>Swift Playgrounds で階層化したノードの SceneKit を試す</b>	<b>72</b>
9.1	コードを書く前に	73
9.2	アプリの初期設定をする	73
9.3	ライト、ノードに紐付いた球を置く	75
9.3.1	シーンのルートにライトを設定する	75

## 目次

---

9.3.2 シーンのルートにノードに紐付いた球を設置する . . . . .	75
9.4 球のチルドノードに球を配置する . . . . .	76
9.5 さらにもう一つ球を設定する . . . . .	77
9.6 太陽に輝きつける . . . . .	77
9.7 アニメーションをつける . . . . .	78
9.7.1 アニメーションさせるノードの構造 . . . . .	78
9.7.2 アニメーションの流れ . . . . .	78
9.7.3 コード . . . . .	79
9.7.4 太陽のテクスチャを調整しアニメーションさせる . . . . .	79
9.8 Xcode の iOS アプリとして実装する . . . . .	80
9.8.1 プロジェクト作成 . . . . .	80
9.8.2 コード編集 . . . . .	81
<b>第 10 章 SceneKit のシーンを ARKit に移植する</b>	<b>82</b>
10.1 コードを編集する . . . . .	83
10.1.1 初期設定 . . . . .	83
10.1.2 前章コードの追加 . . . . .	85
<b>第 11 章 シーン</b>	<b>90</b>
11.1 SCNScene の役割 . . . . .	90
11.2 シーンの初期化 . . . . .	91
11.3 シーンファイルを読み込む . . . . .	91
11.4 シーンの一時停止設定 . . . . .	91
11.5 シーンの背景色、背景画像の設定 . . . . .	92
11.5.1 キューブマップの背景 . . . . .	92
11.5.2 動画の背景 . . . . .	92
11.6 背景画像を照明にするイメージベースドライティングの設定 . . . . .	93
11.6.1 注意点 . . . . .	93
11.7 シーンでの他の項目 . . . . .	93
<b>第 12 章 名前</b>	<b>94</b>
12.1 名前をつける意味 . . . . .	94

## 目次

---

12.2	使用例	94
<b>第 13 章 カテゴリービットマスク</b>		95
13.1	カテゴリービットマスクの振る舞い	95
13.2	カメラでの使用例	95
<b>第 14 章 ノードについて (SCNNode)</b>		97
14.1	移動、回転、拡大縮小	97
14.1.1	移動	97
14.1.2	回転	98
14.1.3	拡大縮小	100
14.2	ピボットの変更	101
14.3	透明度	103
14.4	表示、非表示	103
14.4.1	レンダリングオーダー	104
14.4.2	Movability Hint	104
14.4.3	キャストシャドウ	104
14.5	ノード単位で再生を停止する	104
<b>第 15 章 行列を使用したノードの移動、回転、拡大縮小</b>		106
15.1	SCNMatrix4 の初期設定	106
15.2	行列を使用し拡大縮小させる	107
15.3	行列を使用し移動させる	107
15.4	行列を使用し回転させる	108
15.4.1	X 軸回転	108
15.4.2	Y 軸回転	109
15.4.3	Z 軸回転	109
15.5	もっと簡単に設定したい	110
15.5.1	SCNMatrix4MakeScale	110
15.5.2	SCNMatrix4MakeTranslation	110
15.5.3	SCNMatrix4MakeRotation	110
15.6	簡単に移動、回転、拡大縮小と一緒に使いたい	111

# 目次

---

<b>第 16 章 ジオメトリ</b>	<b>112</b>
16.1 ジオメトリの構成 . . . . .	112
16.2 ビルトインジオメトリの利点 . . . . .	112
16.3 SceneKit で用意されているビルトインジオメトリ . . . . .	113
16.4 無限平面 (SCNFloor) . . . . .	114
16.5 立方体 (SCNBox) . . . . .	115
16.6 カプセル型 (SCNCapsule) . . . . .	118
16.7 円錐 (SCNCone) . . . . .	120
16.8 円柱 (SCNCylinder) . . . . .	121
16.9 平面 (SCNPlane) . . . . .	122
16.10 三角錐 (SCNPyramid) . . . . .	124
16.11 球 (SCNSphere) . . . . .	126
16.12 ドーナツ型 (SCNTorus) . . . . .	128
16.13 チューブ型 (SCNTube) . . . . .	130
16.14 テキスト (SCNText) . . . . .	131
16.15 パスからの図形 (SCNShape) . . . . .	138
<b>第 17 章 カスタムジオメトリ</b>	<b>141</b>
17.1 概要 . . . . .	141
17.2 SceneKit でカスタムジオメトリを作成する流れ。 . . . . .	141
17.3 シーンを作成する . . . . .	142
17.4 三角形の平面を描画する . . . . .	142
17.5 ライトの影響を受けるようにする . . . . .	144
17.6 テクスチャを適応する . . . . .	145
<b>第 18 章 オブジェクトデータをシーンで使用する</b>	<b>148</b>
18.1 読み込むが可能なオブジェクトファイル . . . . .	148
18.2 オブジェクトファイルの適応方法 . . . . .	148
18.3 Blender から DAE ファイルを作成する . . . . .	149
18.4 Blender の DAE エクスポートで使用可能な機能 . . . . .	149
18.5 Blender の DAE エクスポートでの注意点 . . . . .	149
18.6 Mac のターミナルから scn ファイルを作成する . . . . .	150

## 目次

---

<b>第 19 章</b>	<b>アニメーションについて</b>	<b>151</b>
19.1	アニメーションの方法の種類 . . . . .	151
19.2	3 つのアニメーションの違い . . . . .	151
19.2.1	Core Animation と SCNACTION . . . . .	151
19.2.2	SCNTransaction . . . . .	152
19.3	アニメーションの適応範囲とアニメーションの繰り返し処理 . . . . .	152
19.4	アニメーションの同時設定、優先度 . . . . .	152
<b>第 20 章</b>	<b>シーン全体にアニメーションを与える SCNTransaction</b>	<b>153</b>
20.1	SCNTransaction の仕組み . . . . .	153
20.2	SCNTransaction.completionBlock . . . . .	153
20.3	サンプルコード見る . . . . .	153
20.4	SCNTransaction.animationTimingFunction . . . . .	155
20.5	CAMediaTimingFunction 動作例 . . . . .	155
20.5.1	Linear . . . . .	155
20.5.2	EaseIn . . . . .	156
20.5.3	EaseOut . . . . .	156
20.5.4	EaseInEaseOut . . . . .	156
20.5.5	Default . . . . .	157
20.6	その他 . . . . .	157
<b>第 21 章</b>	<b>Core Animation を使用したアニメーション</b>	<b>158</b>
21.1	流れ . . . . .	158
21.2	サンプル . . . . .	158
21.3	Core Animation の始まりと終わりを調べる . . . . .	159
<b>第 22 章</b>	<b>SCNACTION (アクション) を使用したアニメーション</b>	<b>160</b>
22.1	SCNACTION とは . . . . .	160
22.2	アニメーションの再生方法 . . . . .	161
22.3	移動 . . . . .	161
22.3.1	SCNACTION.moveBy - x, y, z で指定 . . . . .	162
22.3.2	SCNACTION.moveBy - SCNVector3 で指定 . . . . .	163

## 目次

---

22.3.3 SCNACTION.moveTo . . . . .	163
<b>22.4 回転 . . . . .</b>	<b>164</b>
22.4.1 SCNACTION.rotateBy - x, y, z で指定 . . . . .	164
22.4.2 SCNACTION.rotateTo - x, y, z で指定 . . . . .	164
22.4.3 SCNACTION.rotateBy - SCNVector3 で指定 . . . . .	164
22.4.4 SCNACTION.rotateTo - SCNVector4 で指定 . . . . .	165
22.4.5 SCNACTION.rotateTo - ShortestUnitArc を指定 . . . . .	165
<b>22.5 拡大縮小 . . . . .</b>	<b>166</b>
22.5.1 SCNACTION.scaleBy . . . . .	166
22.5.2 SCNACTION.scaleTo . . . . .	166
<b>22.6 フェードイン / フェードアウト . . . . .</b>	<b>166</b>
22.6.1 SCNACTION.fadeOut - 徐々に消える . . . . .	167
22.6.2 SCNACTION.fadeIn - 徐々に表示させる . . . . .	167
22.6.3 SCNACTION.fadeOpacity - by . . . . .	167
22.6.4 SCNACTION.fadeOpacity - to . . . . .	167
<b>22.7 表示 / 非表示 . . . . .</b>	<b>168</b>
<b>22.8 逆再生 . . . . .</b>	<b>168</b>
<b>22.9 一時停止 . . . . .</b>	<b>169</b>
22.9.1 一時停止する . . . . .	169
22.9.2 一時停止の振り幅を決める . . . . .	169
<b>22.10 アニメーションのグループ化 . . . . .</b>	<b>170</b>
<b>22.11 アニメーションを順番に再生する . . . . .</b>	<b>170</b>
<b>22.12 アニメーションのリピート . . . . .</b>	<b>171</b>
22.12.1 指定した回数だけリピートする . . . . .	171
22.12.2 無限にリピートする . . . . .	171
<b>22.13 ノードの削除 . . . . .</b>	<b>172</b>
<b>22.14 timingMode と timingFunction の設定 . . . . .</b>	<b>172</b>
<b>第 23 章 SCNACTION でカスタムアニメーションをつくる</b>	<b>174</b>
<b>23.1 customAction の内容 . . . . .</b>	<b>174</b>
<b>23.2 コードサンプル . . . . .</b>	<b>175</b>

# 目次

---

23.3	注意点	175
<b>第 24 章 モーフアニメーション</b>		177
24.1	SCNMorpher の振る舞い	177
24.2	iOS 11 で追加された機能	178
24.3	Blender から書き出された dae ファイルを使用する	179
24.4	Maya から書き出された dae ファイルを使用する	180
24.4.1	サンプルコード	180
24.5	注意点	181
<b>第 25 章 アニメーション機能 <b>SCNAnimation</b> と <b>SCNAnimationPlayer</b></b>		182
25.1	他のアニメーションとの違い	182
25.2	今回の設定の流れ	182
25.3	設定例	182
25.4	SCNAnimation と SCNAnimationPlayer の設定値と関数	186
<b>第 26 章 マテリアル</b>		188
26.1	その前にマテリアルは何をしているの？	188
26.2	ジオメトリへのマテリアルの適応方法	188
26.3	SCNMaterial の基本設定	188
26.4	ライティングモデルとは？	189
26.4.1	各ライティングモデルの説明	189
26.5	ライティングモデルでの負荷	190
26.6	マテリアルの基本設定	190
26.6.1	透明度と値とモード	190
26.6.2	Double sided	191
26.6.3	Lit per pixel	192
26.6.4	Cull Mode	192
26.6.5	Blend Mode	192
26.6.6	Write depth	194
26.6.7	Reads depth	194
26.6.8	Lock ambient with diffuse	194

# 目次

---

<b>第 27 章</b>	<b>マテリアルプロパティ</b>	<b>195</b>
27.1	Physically Based が持つヴィジュアルプロパティ . . . . .	195
27.2	マテリアルプロパティの各パラメーターが持つ値 . . . . .	196
27.2.1	contents . . . . .	196
27.2.2	intensity . . . . .	196
27.3	ヴィジュアルプロパティ (Physically Based) . . . . .	197
27.3.1	Diffuse . . . . .	197
27.3.2	Metalness . . . . .	197
27.3.3	Roughness . . . . .	198
27.3.4	Normal . . . . .	199
27.3.5	Occlusion . . . . .	200
27.3.6	Illumination . . . . .	201
27.3.7	Emission . . . . .	201
27.3.8	Displacement (iOS 11 で追加) . . . . .	202
<b>第 28 章</b>	<b>テクスチャ</b>	<b>204</b>
28.1	SCNMaterialProperty のテクスチャ関連設定 . . . . .	204
28.2	contentsTransform . . . . .	204
28.3	wrapS、wrapT . . . . .	205
28.3.1	Clamp . . . . .	205
28.3.2	Repeat . . . . .	205
28.3.3	Mirror . . . . .	205
28.4	フィルター . . . . .	205
28.4.1	フィルタリング設定 . . . . .	205
28.4.2	minificationFilter . . . . .	206
28.4.3	magnificationFilter . . . . .	206
28.4.4	mipFilter . . . . .	206
28.4.5	maxAnisotropy . . . . .	206
28.5	writesToDepthBuffer . . . . .	206
28.6	readsFromDepthBuffer . . . . .	206
28.7	colorBufferWriteMask . . . . .	207

# 目次

---

28.8	mappingChannel	207
<b>第 29 章 キューブマップ</b>		<b>208</b>
29.1	キューブマップの画像配置	208
29.2	キューブマップ使用できる画像サイズ	208
29.3	Vertical strip	208
29.4	Horizontal strip	210
29.5	Spherical projection	210
29.6	Array of six images	211
<b>第 30 章 カメラ</b>		<b>212</b>
30.1	カメラの設定	212
30.2	Projection Type	212
30.3	focalLength、fieldOfView、sensorHeight	212
30.4	Z Clipping	213
30.5	被写界深度	213
30.6	モーションブラー	213
30.7	iOS 11 追加されたが設定がわからないもの	214
<b>第 31 章 ライト</b>		<b>215</b>
31.1	SCNLight の種類	215
31.2	SCNLight の制限	215
31.3	SCNLight の注意点	216
31.4	SCNLight の共通設定	216
31.4.1	Type	216
31.4.2	Mode	217
31.4.3	Color	217
31.4.4	Intensity	217
31.4.5	temperature	217
31.4.6	その他	219
31.5	Omni Light と Attenuation 設定	219
31.5.1	Attenuation	219

# 目次

---

31.6	Ambient Light . . . . .	220
31.7	Spot Light とキャストシャドウ（シャドウマップ） . . . . .	220
31.7.1	Attenuation 以外の個別設定 . . . . .	221
31.7.2	Casts Shadows . . . . .	222
31.8	Directional Light . . . . .	223
31.9	IES Light . . . . .	223
31.9.1	注意点 . . . . .	226
31.10	Light Probe . . . . .	226
31.10.1	概要 . . . . .	226
31.10.2	Light Probe の制限 . . . . .	226
<b>第 32 章</b>	<b>物理アニメーションの概要</b>	<b>227</b>
32.1	物理アニメーションで設定するもの . . . . .	227
32.1.1	その他 . . . . .	227
32.2	物理アニメーションの種類 . . . . .	228
32.2.1	Static . . . . .	228
32.2.2	Dynamic . . . . .	228
32.2.3	Kinematic . . . . .	228
32.3	処理負荷 . . . . .	229
32.4	物理アニメーションを行う . . . . .	229
<b>第 33 章</b>	<b>物理アニメーションの共通設定</b>	<b>232</b>
33.1	質量、抵抗、電荷 . . . . .	232
33.1.1	Mass . . . . .	232
33.1.2	Friction . . . . .	232
33.1.3	Restitution . . . . .	233
33.1.4	Rolling friction . . . . .	233
33.1.5	Damping . . . . .	233
33.1.6	Angular damping . . . . .	233
33.1.7	Charge . . . . .	233
33.1.8	Gravity (isAffectedByGravity) . . . . .	234
33.1.9	Resting (allowsResting) . . . . .	234

## 目次

---

33.2	質量、抵抗、電荷の設定例 . . . . .	234
33.3	速さやその力 . . . . .	235
33.3.1	Linear velocity . . . . .	235
33.3.2	Angular velocity . . . . .	235
33.3.3	Linear factor . . . . .	235
33.3.4	Angular factor . . . . .	235
33.4	速さやその力の設定例 . . . . .	235
33.5	慣性モーメント . . . . .	236
33.6	慣性モーメントの設定例 . . . . .	236
33.7	力、トルク、衝動（力積、インパルス） . . . . .	237
33.7.1	動作 . . . . .	237
33.7.2	単位（通常） . . . . .	237
33.7.3	単位（ <code>asImpulse</code> が <code>true</code> の場合） . . . . .	237
33.8	力、トルク、衝動（力積、インパルス）の設定例 . . . . .	237
33.9	物理アニメーションの全体設定 . . . . .	238
33.9.1	<code>gravity</code> . . . . .	238
33.9.2	<code>speed</code> . . . . .	238
33.9.3	<code>timeStep</code> . . . . .	239
33.9.4	<code>updateCollisionPairs()</code> . . . . .	239
<b>第 34 章</b>	<b>物理アニメーションのカテゴリビットマスク</b>	<b>240</b>
34.1	各ビットマスクについて . . . . .	240
34.2	<code>Category mask</code> . . . . .	240
34.3	<code>Collision mask</code> . . . . .	240
34.4	<code>Contact mask</code> . . . . .	241
34.5	<code>Category mask</code> と <code>Collision mask</code> の使用例 . . . . .	241
34.5.1	球 . . . . .	242
34.5.2	板 . . . . .	242
34.5.3	床（ <code>SCNFloor</code> ） . . . . .	242
<b>第 35 章</b>	<b>物理アニメーションの接触処理</b>	<b>243</b>
35.1	注意点 . . . . .	243

# 目次

---

35.2	SCNPhysicsContact で取得できるもの . . . . .	243
35.3	SCNPhysicsContactDelegate の命令 . . . . .	244
35.4	設定手順 . . . . .	244
35.5	SCNPhysicsContactDelegate を設定する . . . . .	244
<b>第 36 章</b>	<b>物理フィールドの概要</b>	<b>248</b>
36.1	物理フィールドの使用例 . . . . .	248
36.2	SceneKit で用意されている物理フィールド . . . . .	249
36.2.1	halfExtent の設定 . . . . .	250
36.2.2	scope の outside . . . . .	250
36.2.3	フィールド効果の減衰 . . . . .	250
<b>第 37 章</b>	<b>各物理フィールドについて</b>	<b>251</b>
37.1	Drag Field . . . . .	251
37.1.1	Strength . . . . .	251
37.2	Vortex Field . . . . .	251
37.2.1	Strength . . . . .	252
37.2.2	Direction (SCNVector3) . . . . .	252
37.2.3	Offset (SCNVector3) . . . . .	252
37.3	Radial Gravity Field . . . . .	252
37.3.1	Strength . . . . .	253
37.3.2	Offset (SCNVector3) . . . . .	253
37.4	Linner Gravity Field . . . . .	253
37.4.1	Strength . . . . .	254
37.4.2	Direction (SCNVector3) . . . . .	254
37.5	Noise Field . . . . .	254
37.5.1	Strength . . . . .	255
37.5.2	smoothness (CGFloat) . . . . .	255
37.5.3	speed (CGFloat) . . . . .	255
37.6	Turbulence Field . . . . .	255
37.6.1	Strength . . . . .	256
37.6.2	smoothness (CGFloat) . . . . .	256

# 目次

---

37.6.3 speed (CGFloat)	256
37.7 Spring Field	256
37.7.1 Strength	257
37.7.2 Offset (SCNVector3)	257
37.8 Electric Field	257
37.8.1 Strength	258
37.8.2 Offset (SCNVector3)	258
37.9 Magnetic Field	258
37.9.1 Strength	259
37.10 パーティクルを物理フィールドでも動くように設定する	259
37.11 設定例	259
<b>第 38 章 パーティクルシステム</b>	<b>261</b>
38.1 SCNParticleSystem の設定方法の種類	262
38.2 SceneKit Particle System ファイル (scnp) をシーンに適応する	262
38.3 パーティクルシステムの簡単な流れ	263
38.3.1 パーティクルの画像設定	263
38.3.2 パーティクルの寿命	263
38.3.3 エミッタ	263
38.3.4 バリエーション	263
38.3.5 動き	264
38.3.6 他の機能	264
<b>第 39 章 コンストRAINT (制約)</b>	<b>265</b>
39.1 コンストRAINTの種類	265
39.2 コンストRAINTの使用例	265
39.3 SCNBillboardConstraint	266
39.3.1 freeAxis の設定値	266
39.4 SCNLookAtConstraint	266
39.5 SCNDistanceConstraint (iOS 11 で追加)	267
39.6 SCNAvoidOccluderConstraint (iOS 11 で追加)	267
39.7 SCNAccelerationConstraint (iOS 11 で追加)	268

## 目次

---

39.8	SCNSliderConstraint (iOS 11 で追加) . . . . .	268
39.9	SCNReplicatorConstraint (iOS 11 で追加) . . . . .	269
<b>第 40 章</b>	<b>SpriteKit のシーンをテクスチャとして使用する</b>	<b>270</b>
40.1	SKTexture をテクスチャとして使用する . . . . .	270
40.2	SpriteKit のシーンをテクスチャとして使用する . . . . .	271
	40.2.1 SpriteKit の SKVideoNode を使用し動画のテクスチャを使用する . . . . .	271
<b>第 41 章</b>	<b>HDR</b>	<b>274</b>
41.1	HDR の使用方法 . . . . .	274
41.2	Average Gray . . . . .	274
41.3	White Point . . . . .	275
41.4	Adaptation . . . . .	275
	41.4.1 Auto Adapt . . . . .	275
	41.4.2 Speed Factor . . . . .	275
41.5	Exposure . . . . .	275
	41.5.1 Minimum、Maximum . . . . .	275
	41.5.2 Offset . . . . .	276
41.6	Bloom . . . . .	276
	41.6.1 Intensity . . . . .	276
	41.6.2 Threshold . . . . .	276
	41.6.3 Blur Radius . . . . .	276
41.7	サンプルコード . . . . .	277
<b>第 42 章</b>	<b>ポストプロセス</b>	<b>278</b>
42.1	Vignetting . . . . .	278
42.2	Color Fringe . . . . .	279
42.3	Saturation . . . . .	280
42.4	Contrast . . . . .	281
42.5	モーションブラー . . . . .	282
<b>第 43 章</b>	<b>被写界深度</b>	<b>284</b>
43.1	概要 . . . . .	284

# 目次

---

43.2	焦点距離と F 値	284
43.3	使用例	288
<b>第 44 章 フォグ（霧）</b>		<b>290</b>
44.1	フォグの効果の原理	290
44.2	設定値	290
44.3	設定例	290
<b>第 45 章 シーンの Point of View</b>		<b>292</b>
45.1	コードでの設定方法	292
45.2	カメラの適応について	293
<b>第 46 章 CameraController</b>		<b>294</b>
46.1	インタラクションモード	294
46.1.1	fly	294
46.1.2	orbitTurntable	295
46.1.3	orbitAngleMapping	295
46.1.4	orbitCenteredArcball	295
46.1.5	orbitArcball	295
46.1.6	pan	295
46.1.7	truck	295
46.2	CameraController 設定値	295
46.2.1	interactionMode	296
46.2.2	inertiaEnabled	296
46.2.3	inertiaFriction	296
46.2.4	isInertiaRunning	296
46.2.5	minimumVerticalAngle、maximumVerticalAngle minimumHorizontalAngle、maximumHorizontalAngle	297
46.2.6	pointOfView	297
46.2.7	target	297
46.2.8	automaticTarget	297
46.2.9	worldUp	297

## 目次

---

46.2.10 delegate . . . . .	297
46.3 CameraController のメソッド . . . . .	298
46.3.1 clearRoll() . . . . .	298
46.3.2 stopInertia() . . . . .	298
46.3.3 translateInCameraSpaceBy(x: Float, y: Float, z: Float) . . . . .	298
46.3.4 frameNodes([SCNNode]) . . . . .	298
46.3.5 rotateBy(x: Float, y: Float) . . . . .	298
46.3.6 dolly(by: Float, onScreenPoint: CGPoint, viewport: CGSize) . . . . .	299
46.3.7 dolly(toTarget: Float) . . . . .	299
46.3.8 roll(by: Float, aroundScreenPoint: CGPoint, viewport: CGSize) . . . . .	299
46.3.9 rollAroundTarget(Float) . . . . .	299
46.3.10 インタラクションが起こった際に動作させる関数 . . . . .	299
46.4 SCNCameraControlConfiguration . . . . .	299
46.4.1 allowsTranslation . . . . .	300
46.4.2 autoSwitchToFreeCamera . . . . .	300
46.4.3 flyModeVelocity . . . . .	300
46.4.4 panSensitivity、rotationSensitivity、truckSensitivity . . . . .	300
46.5 ARKit での CameraController の振る舞いについて . . . . .	300
<b>第 47 章 Screen Space Ambient Occlusion (SSAO)</b> . . . . .	<b>301</b>
47.1 Screen Space Ambient Occlusion の仕組み . . . . .	301
47.2 SSAO の設定方法 . . . . .	301
47.3 注意点 . . . . .	302
47.4 設定値 . . . . .	302
47.4.1 Intensity . . . . .	302
47.4.2 Radius . . . . .	302
47.4.3 Bias . . . . .	302
47.4.4 Depth Threshold . . . . .	302
47.4.5 Normal Threshold . . . . .	303
47.5 コードでの記述 . . . . .	303
<b>第 48 章 効果音や BGM の再生</b> . . . . .	<b>304</b>

## 目次

---

48.1	音を再生する種類	304
48.2	SCNAudioSource	304
48.3	SCNAudioPlayer	305
48.4	SCNNode で SCNAudioPlayer を扱うための機能	306
48.5	SCNAudioPlayer 使用をして再生する	307
48.6	SCNACTION で SCNAudioSource を設定する	307
48.7	注意点	308
<b>第 49 章</b>	<b>テッセレーションとサブディビジョンサーフェイス (A9 以降の端末のみ)</b>	<b>309</b>
49.1	テッセレーション	309
49.2	テッセレーションの設定値	310
49.3	テッセレーションの設定方法	313
49.4	サブディビジョンサーフェイス	314
49.5	サブディビジョンサーフェイスの設定方法	315
49.6	頂点やエッジに対してクリーズで適応度合いの変更する	315
49.7	サブディビジョンサーフェイス設定での注意点	315
<b>第 50 章</b>	<b>ノードの最適化</b>	<b>316</b>
50.1	Clone (複製)	316
50.1.1	Clone 使用例	316
50.1.2	Clone での注意点	316
50.2	階層化しているノードを一つにまとめる	317
50.2.1	注意点	317
50.2.2	使用例	318
50.3	他のシーンファイルを参照として配置する	318
<b>第 51 章</b>	<b>iOS 11 で追加されたノードの機能</b>	<b>320</b>
51.1	SIMD	320
51.2	プロパティ	321
51.2.1	既存のものに SIMD が使えるようになったもの	321
51.2.2	SceneKit のプロパティで追加されたもの	321
51.3	タイププロパティ	322

## 目次

---

51.4	メソッド . . . . .	323
51.4.1	convertVector(SCNVector3, to: SCNNNode?), convertVector(SCNVector3, from: SCNNNode?) . . . . .	323
51.4.2	localRotate(by: SCNQuaternion) . . . . .	323
51.4.3	localTranslate(by: SCNVector3) . . . . .	323
51.4.4	look(at: SCNVector3)、look(at: SCNVector3, up: SCNVector3, localFront: SCNVector3) . . . . .	324
51.4.5	rotate(by: SCNQuaternion, aroundTarget: SCNVector3) . . . . .	324
51.4.6	setWorldTransform(SCNMatrix4) . . . . .	324
51.5	メソッド SIMD 版 . . . . .	324
51.5.1	simdConvertPosition(simd_float3, from: SCNNNode?), simdCon- vertPosition(simd_float3, to: SCNNNode?) . . . . .	325
第 52 章	ジオメトリに <b>Core Image</b> のフィルターを使用する	326
52.1	概要 . . . . .	326
52.2	実行例 . . . . .	326
第 53 章	統計情報とデバッグオプション	328
53.1	統計情報 . . . . .	328
53.1.1	詳細表示用の「+」ボタン . . . . .	329
53.1.2	1 フレームでのレンダリングの流れ . . . . .	329
53.2	デバッグオプション . . . . .	331
53.2.1	showBoundingBoxes . . . . .	331
53.2.2	showLightInfluences . . . . .	331
53.2.3	showLightExtents . . . . .	331
53.2.4	showPhysicsFields . . . . .	332
53.2.5	showPhysicsShapes . . . . .	332
53.2.6	showWireframe . . . . .	332
53.2.7	renderAsWireframe (iOS 11) . . . . .	332
53.2.8	showCameras (iOS 11) . . . . .	332
53.2.9	showConstraints (iOS 11) . . . . .	332
53.2.10	showCreases (iOS 11) . . . . .	333

## 目次

---

53.2.11showSkeletons (iOS 11) . . . . .	333
53.2.12設定例 . . . . .	333
<b>第 54 章 カスタムシェーダー</b>	<b>334</b>
54.1 SCNShadable の shaderModifiers とは? . . . . .	334
54.2 使用方法 . . . . .	335
54.3 設定例 . . . . .	335
54.4 Geometry modifier . . . . .	335
54.5 Surface modifier . . . . .	336
54.6 Lighting model modifier . . . . .	336
54.7 Fragment modifier . . . . .	337
54.8 その他 . . . . .	338
<b>macOS アプリでのクリックイベント</b>	<b>339</b>
<b>編集後記</b>	<b>341</b>

# 第1章

## 3DCGとは？

---

### 1.1 SceneKitを始める前に 3DCGで必要なものとは？

映画、CM、アニメーションなどの映像作品で使われている3DCGですが、仮想的な3次元の空間で映像を作成していますが、現実世界の映像収録と同じような最低限必要な構成があります。

#### 1.1.1 カメラ

映像を撮るため、必ずカメラで何かを写すことになります。

#### 1.1.2 ライト

現実世界には外に出ると太陽が光源となります。光が遮断された部屋などに入ると真っ暗になります。そのため、光がない空間でカメラの撮る映像は暗闇となり何も映らなくなります。

#### 1.1.3 オブジェクト

カメラとライトがあっても、撮る対象がない場合は何も映っていないことと同じです。背景や登場する何かがないと、永遠に同じような画面の映像を観ることになるので、何かを設置することになります。

#### 1.1.4 マテリアル

こちらのみ現実世界と異なり、3DCG特有のもので、物体の表面に何らかの素材の設定を行うことが必要になります。現実の世界ではプラスチックはプラスチックの外観を初めから持っていますが、仮想空間である3DCGではその設定してあげなければいけません。

### 1.2 リアルタイムレンダリングとオフラインレンダリング

3DCGの映像には2種類の表示方法があり、1つは端末のGPUを使用して即時に表示し続けるリアルタイムレンダリング、もう一つは事前に時間をかけて計算して表示するものをオフラインレンダリングと呼ばれています。

ちなみに、3DCGで言うレンダリングとは、仮想空間である3次元空間の情報から2次元の画像に変換することを指します。人の目は水晶体というレンズを使っていますし、デバイスの画面はピクセルの密集した2Dなので、最終的には2次元の画像を描く必要があるためです。

### 1.3 ワールド座標とローカル座標

3DCGの仮想空間ではワールド座標とローカル座標というものが存在します。

ワールド座標とはシーンや空間で決められた固定されている座標で、ローカル座標はキャラクタやライトカメラなどオブジェクトが持っている座標です。

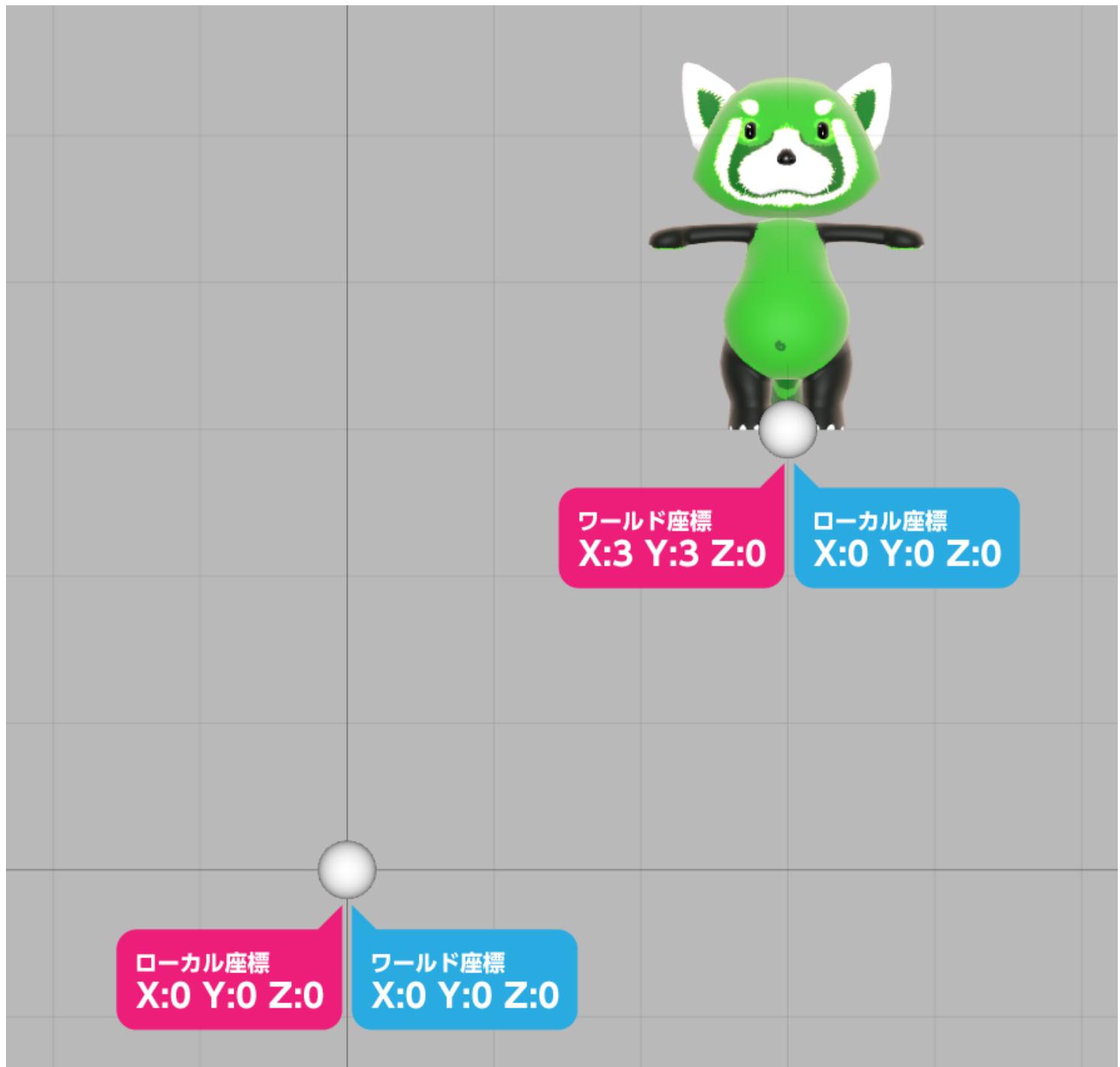


図 1.1 ローカル座標とワールド座標

現実で例えるなら、ワールド座標は緯度経度のように世界で位置が決まっている距離で、ローカル座標は家を起点として考えた会社までの距離です。  
(あまり上手い説明になっていない……)

# 第2章

## Swift Playgrounds for iPad の使い方

### 2.1 アプリの入手

App Store から検索して Swift Playgrounds<sup>\*1</sup> をダウンロードしましょう。

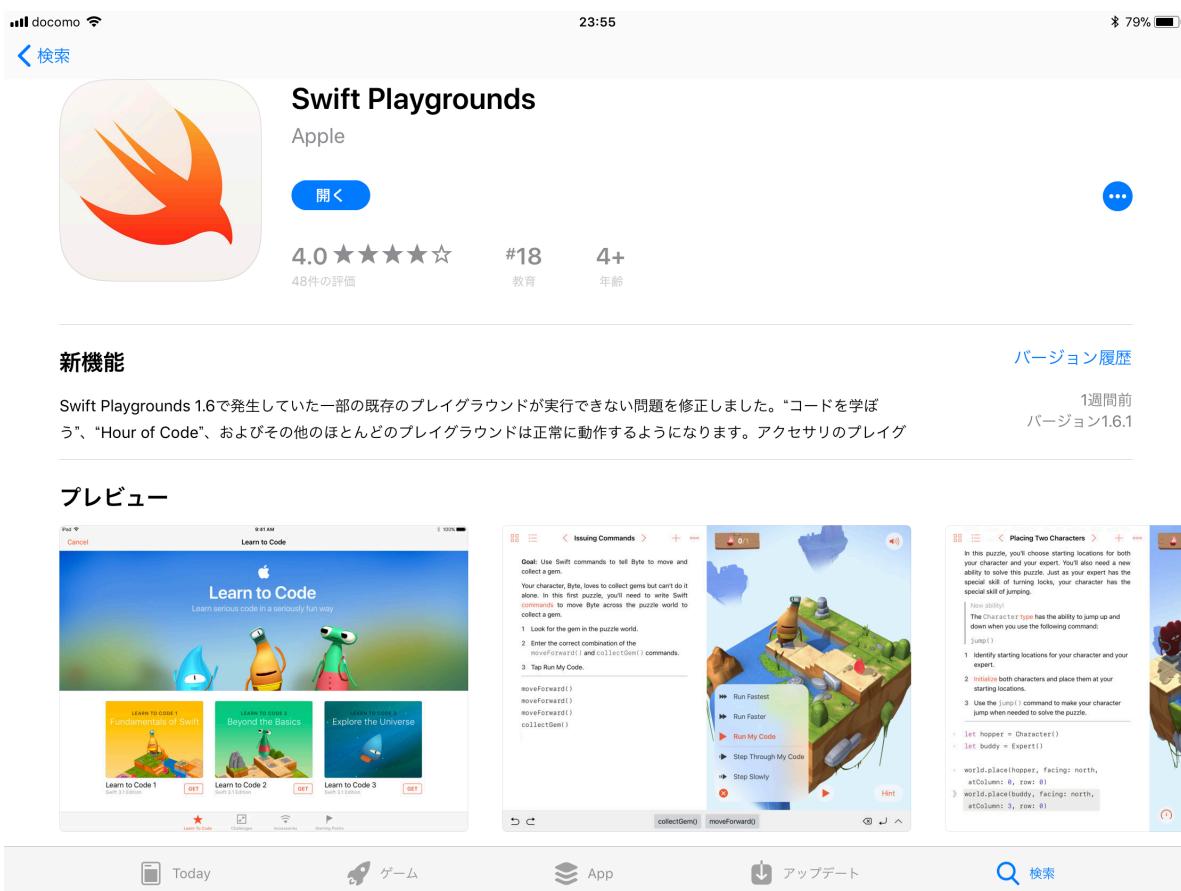


図 2.1 App Store

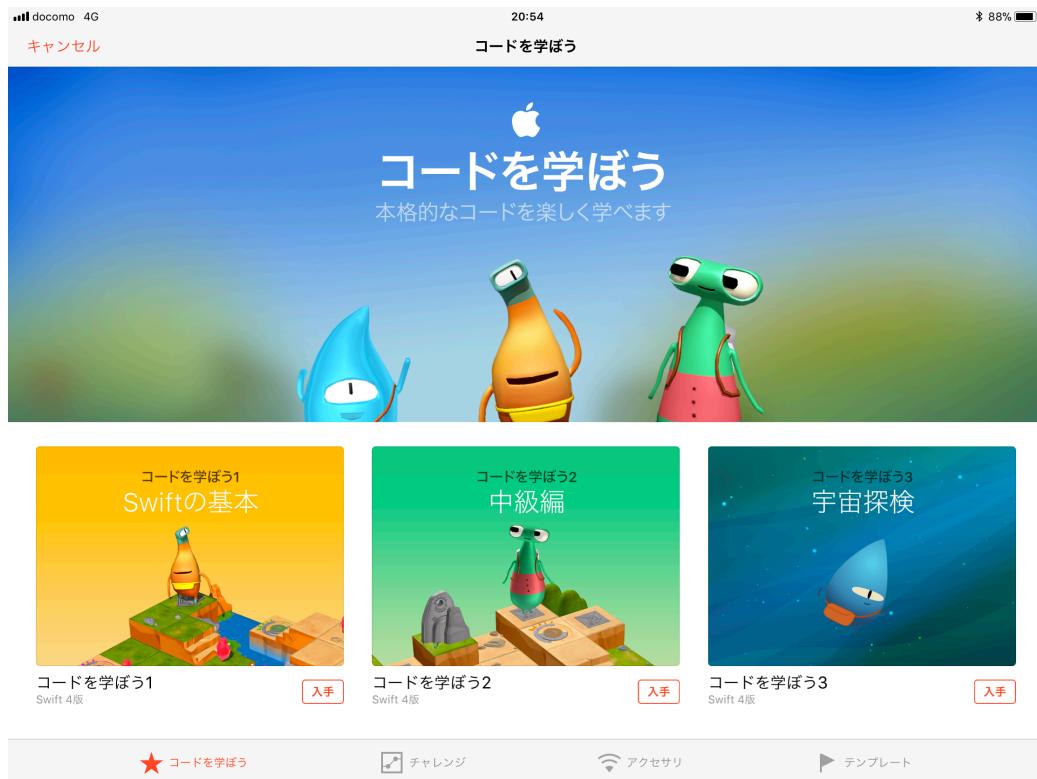
\*1 <https://itunes.apple.com/jp/app/swift-playgrounds/id908519492?mt=8>

### 2.2 空白（Blank）の Playground ページのつくり方

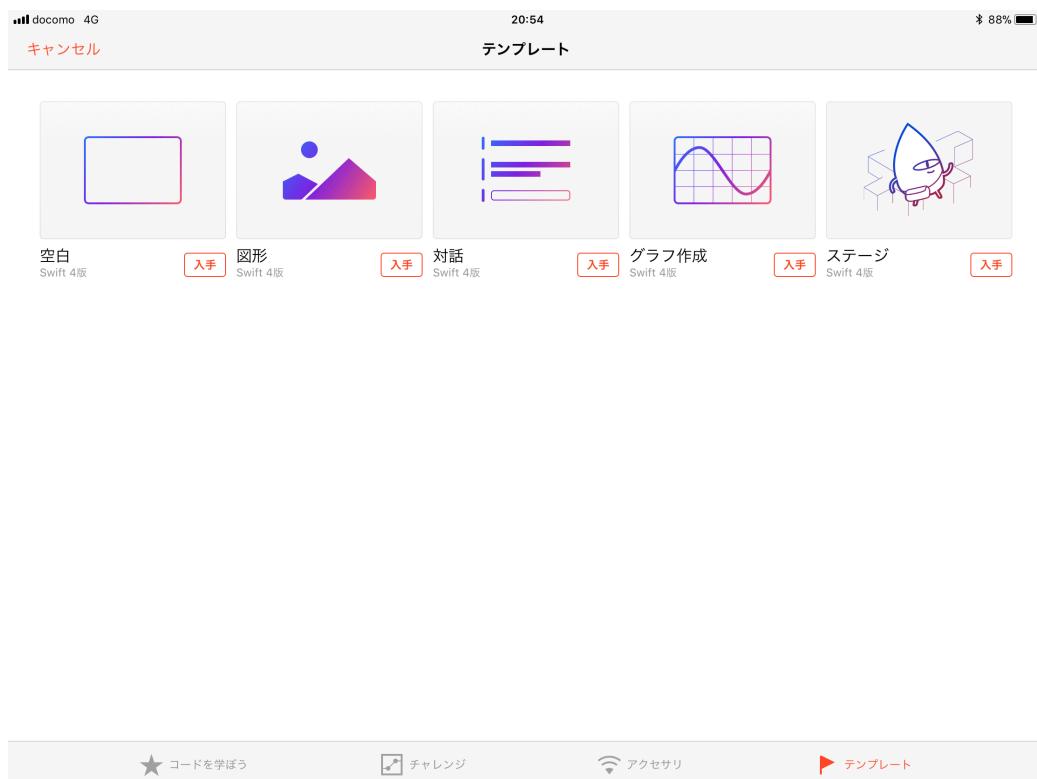


アプリを起動完了すると、右上「+」のアイコンの「新しいプレイグラウンド」をタップ。

## 第2章 Swift Playgrounds for iPad の使い方



今回は右下の「テンプレート」を選択。

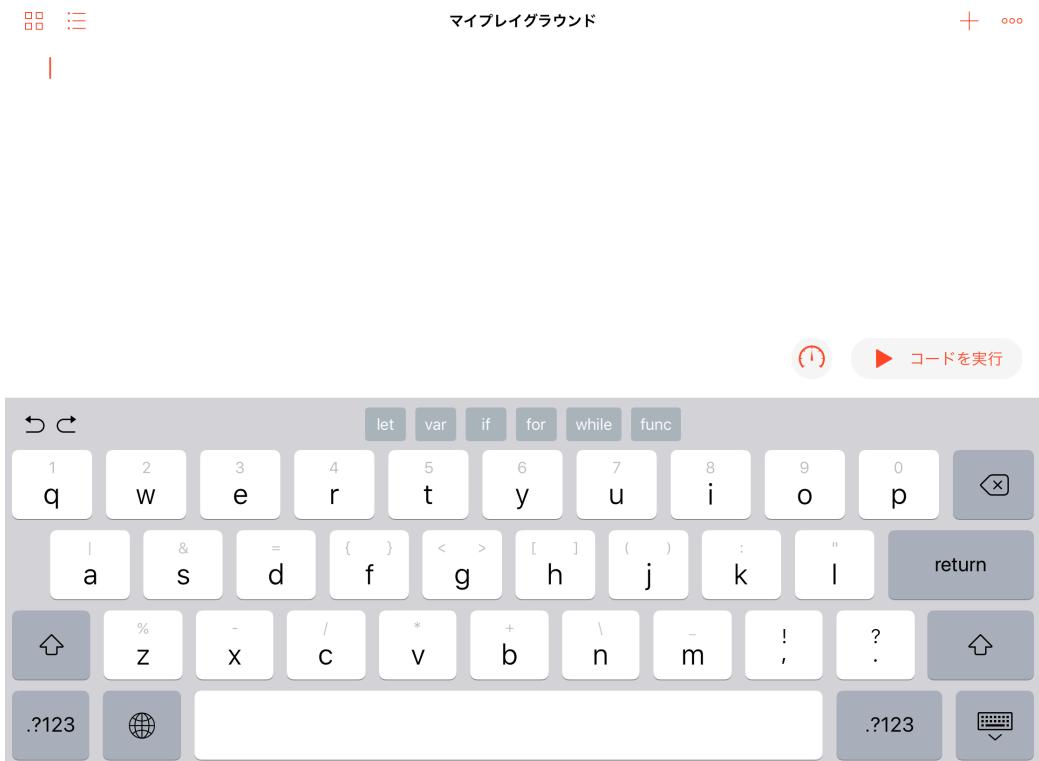


テンプレート選択画面になるので「空白」を選択。

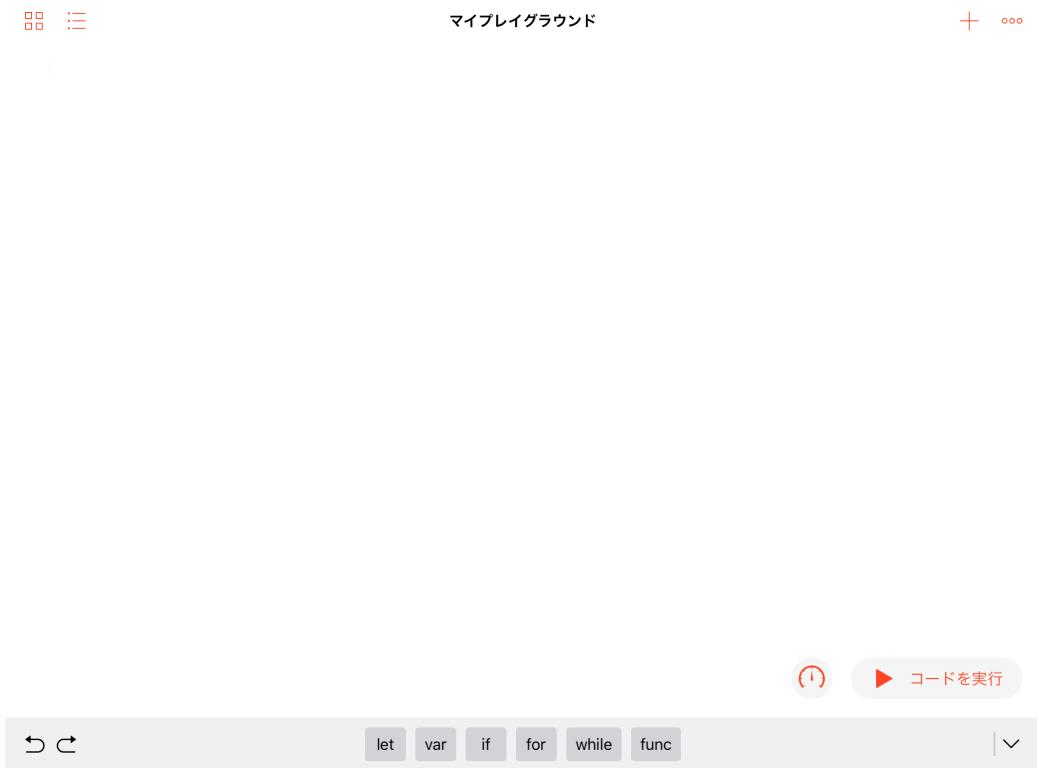
## 第2章 Swift Playgrounds for iPad の使い方



「入手」を押すと何もない Playground ページが作成されます。



スマートキーボードや Bluetooth のキーボードを使用している場合は UI が変わります。



### 2.3 UI

今回は横画面で行いますが、縦でも使用することができます。

左上

- ホームに戻る
- プレイグラウンドページ編集（本書では使用しません）

右上

- リソースとコードスニペット
- ツール

右下

- ステップ実行への変更
- コード実行

### キーボードのアクセサリビュー

- ・ アンドゥ、リドゥ
- ・ キーボードを閉じる（外部キーボードのみ）

## 2.4 プレイグラウンドページ編集

編集ボタンを押すとプレイグラウンドの編集ができます。複数のページでコードを記述することができます。



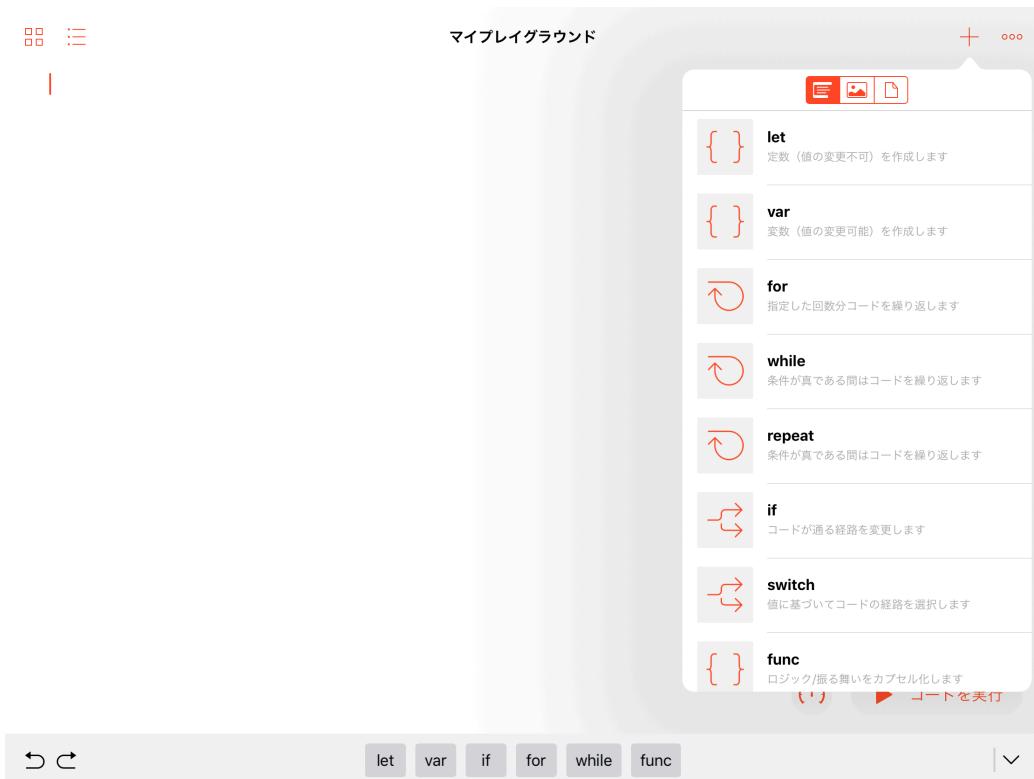
## 2.5 リソースとコードスニペット

コードスニペットや外部ファイルを読み込むことができます。

### 2.5.1 コードスニペット

標準的な命令が列挙されタップするとコードが書かれます。

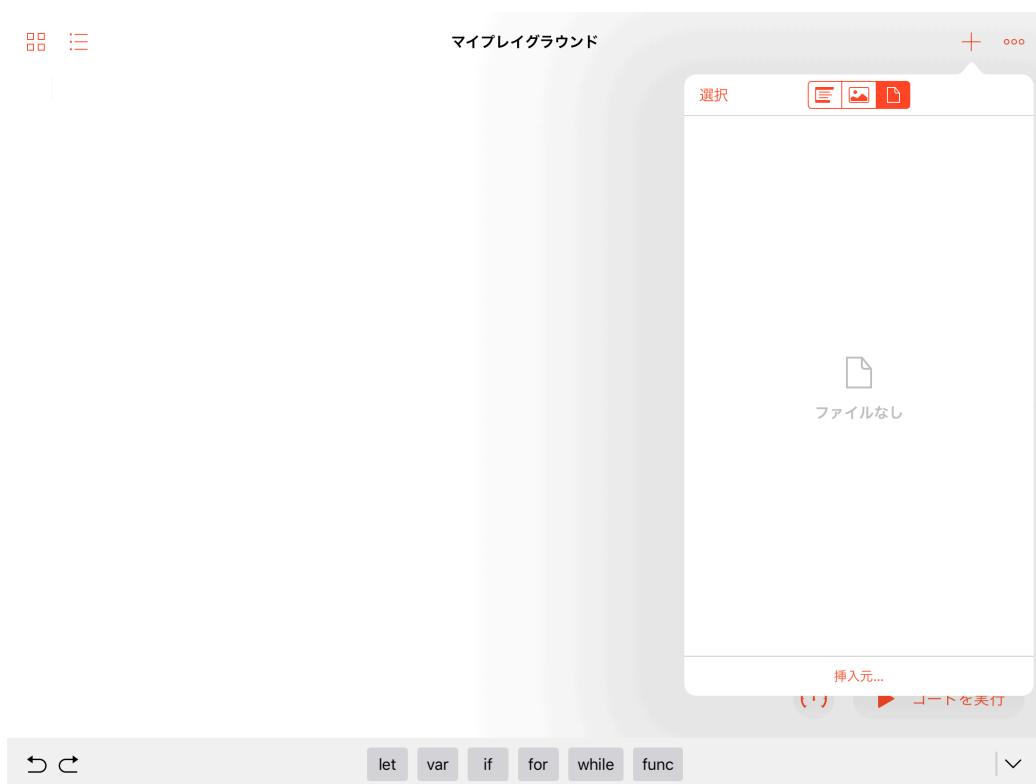
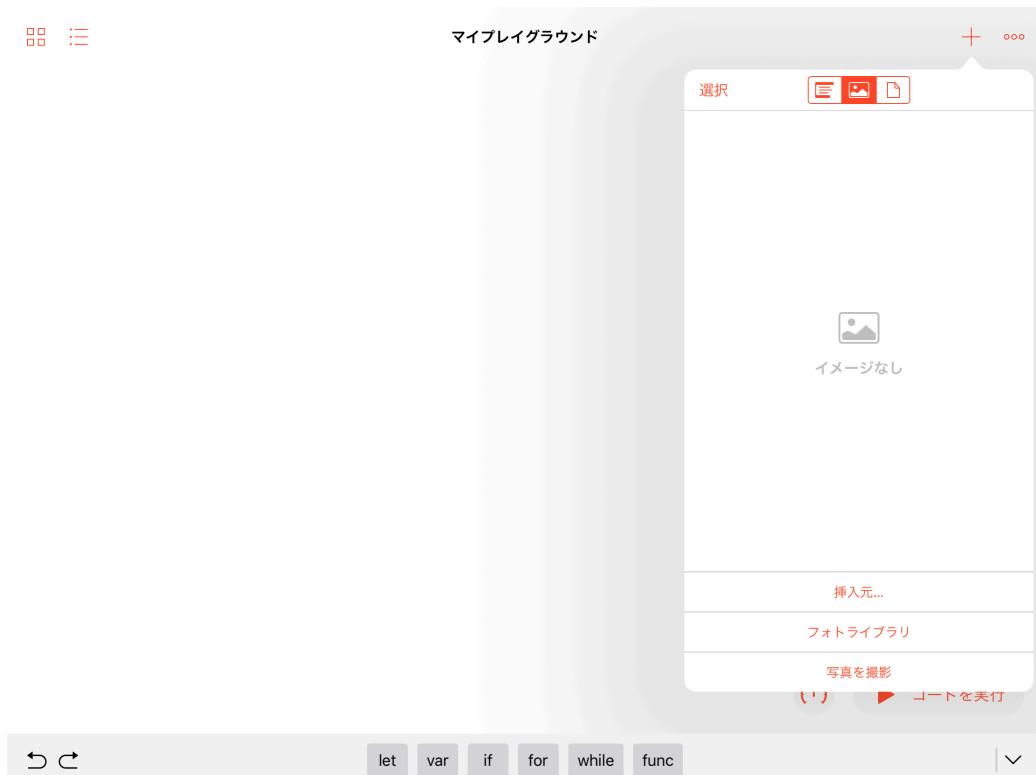
## 第2章 Swift Playgrounds for iPad の使い方



### 2.5.2 リソース

写真アプリのフォトライブラリか、iCloud Drive からリソースファイルを取得します。SceneKit のシーンファイル、Storyboard、CoreML の .mlmodelc など iOS 関連づけられていないファイルも読み込むことができます。

## 第2章 Swift Playgrounds for iPad の使い方



### 2.6 ツール

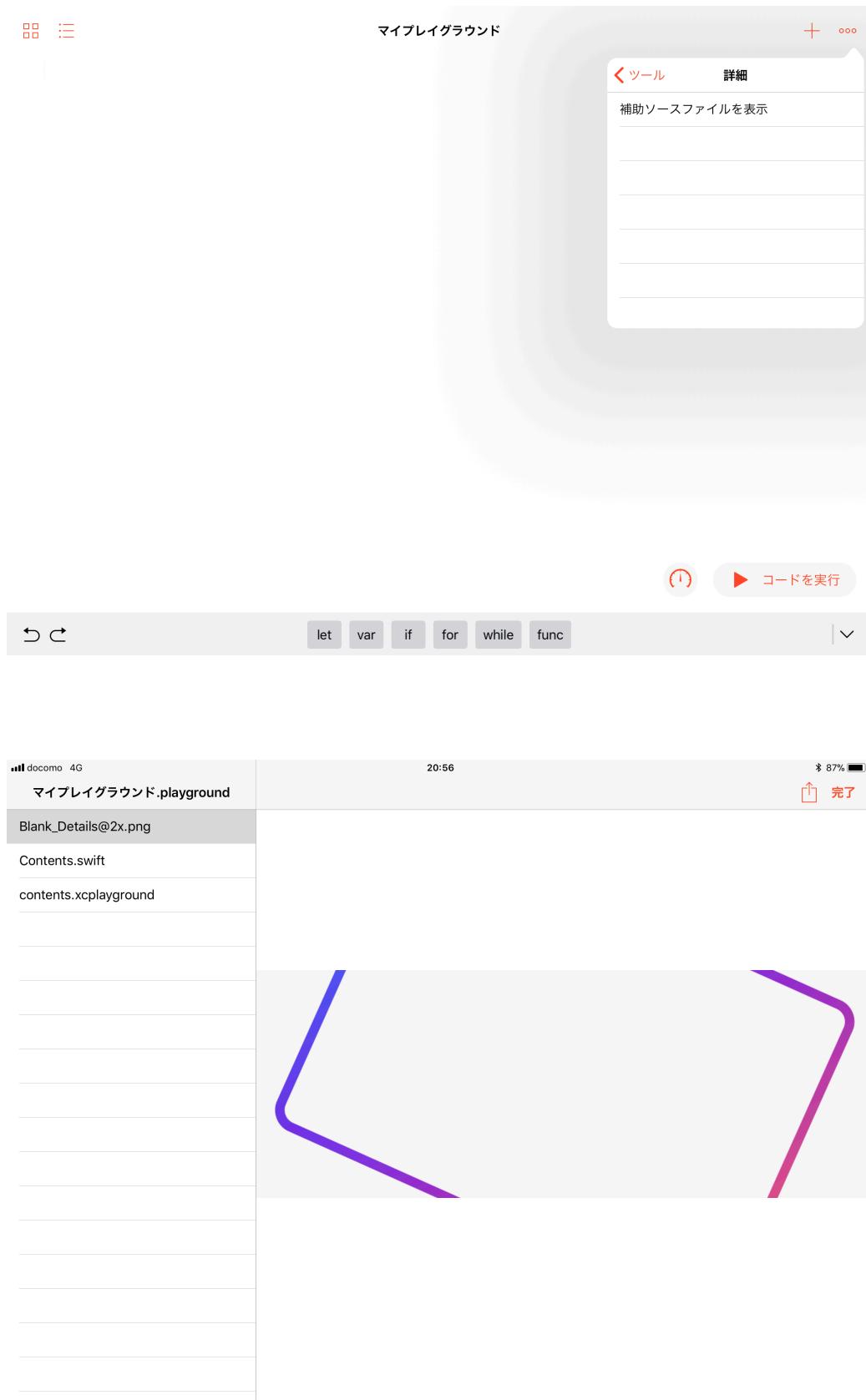
- 写真を撮影
- PDF を作成
- ムービーを収録
- ライブでブロードキャスト



#### 2.6.1 詳細

補助ソースファイルの表示から編集はできませんが内部で使用しているファイルを見ることができます。

## 第2章 Swift Playgrounds for iPad の使い方



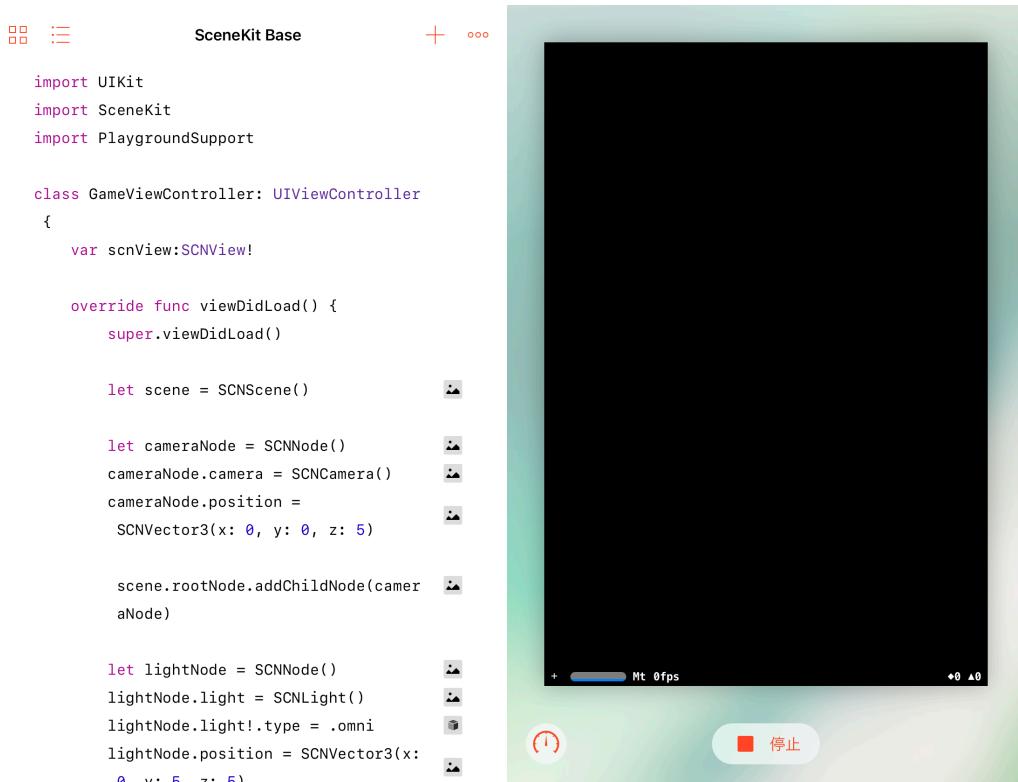
### 2.7 ステップ実行への変更

「コード実行」を押すと即ビルドし始め実行始めるまで止めることができませんが、実行までの速度を変更することができ、コードの動きを見るすることができます。



### 2.8 コード実行

タップするとコード実行し、ライブビューに実行結果が表示されます。ライブビューは右からのスワイプ（縦の場合は下から）で表示させることができます。



## 2.9 アンドウ、リドウ

ノートアプリのようなアンドウでやり直しができます。

## 2.10 キーボードを閉じる（外部キーボードのみ）

外部キーボードを使用している場合、キーボードの表示を完全に消すことができます。

## 2.11 コードエラー

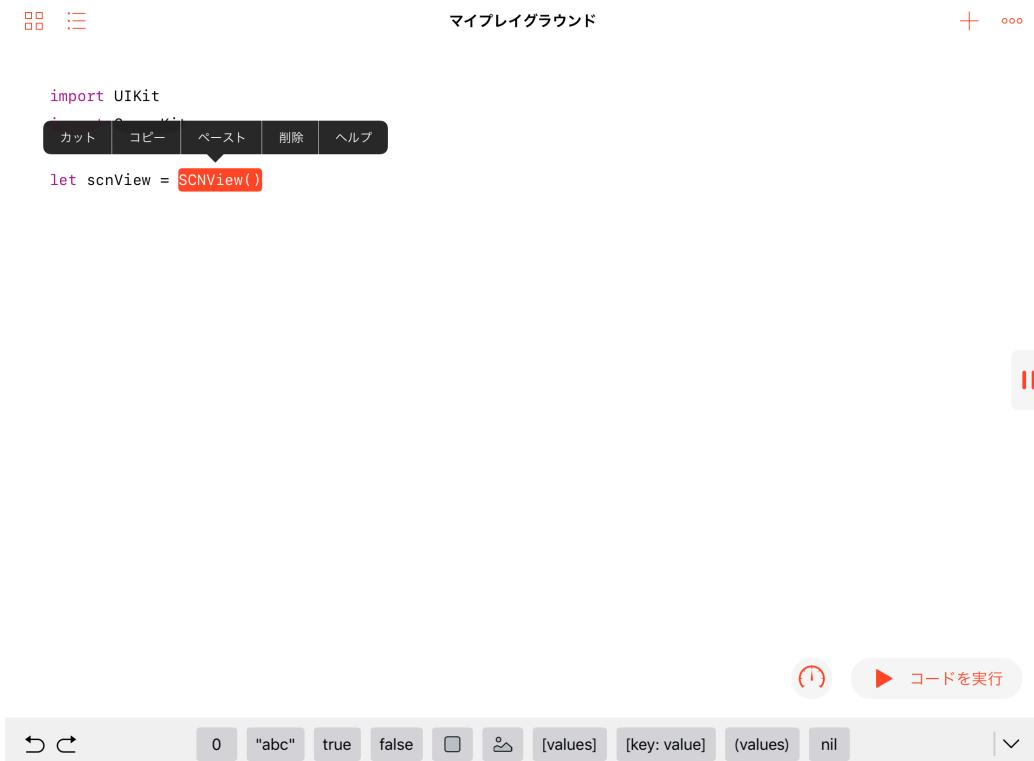
エラーがある場合は右に丸が表示されタップすると、英語ですがエラー内容が表示されます。



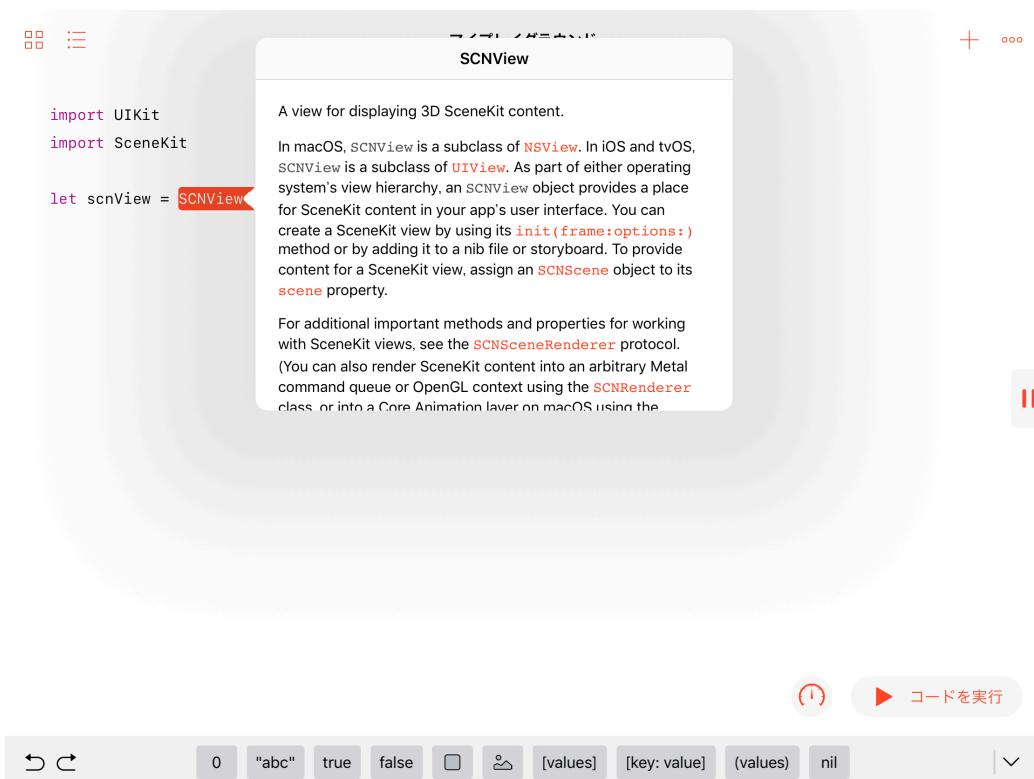
### 2.12 ヘルプ

記述されているものをタップすると、ヘルプのボタンが表示されます。

## 第2章 Swift Playgrounds for iPad の使い方

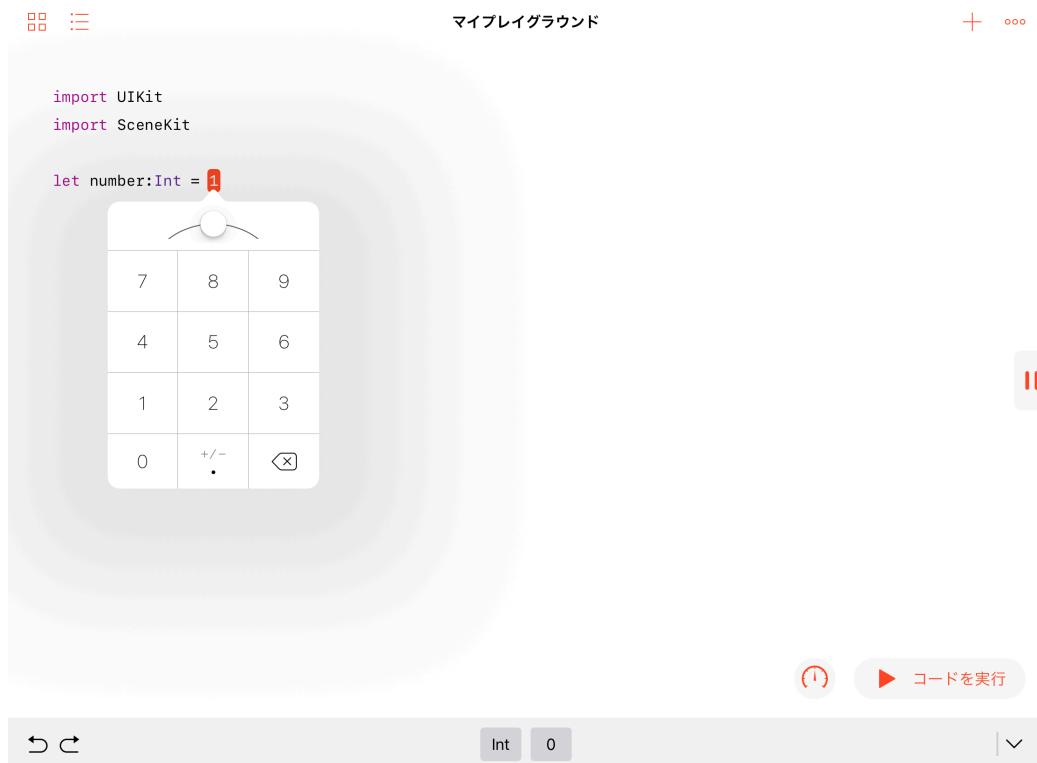


ヘルプをタップするとその命令の定義が表示されます。



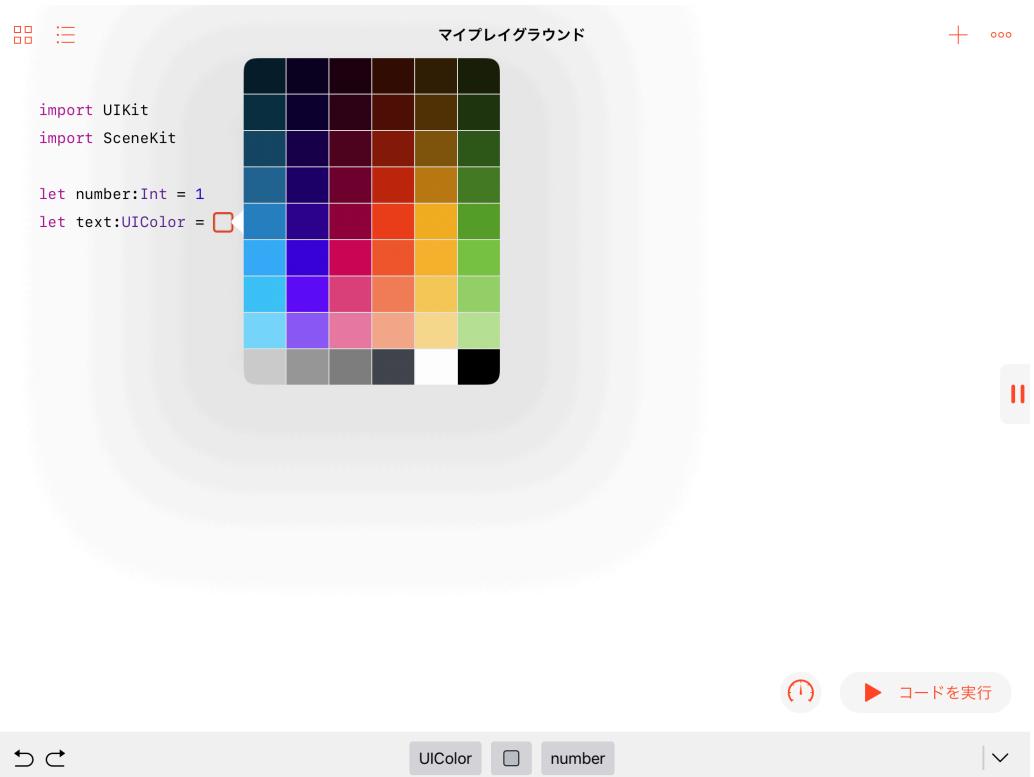
### 2.13 ナンバーパット

コード内の数値を選択すると以下の表示なり通知を選択することができます。



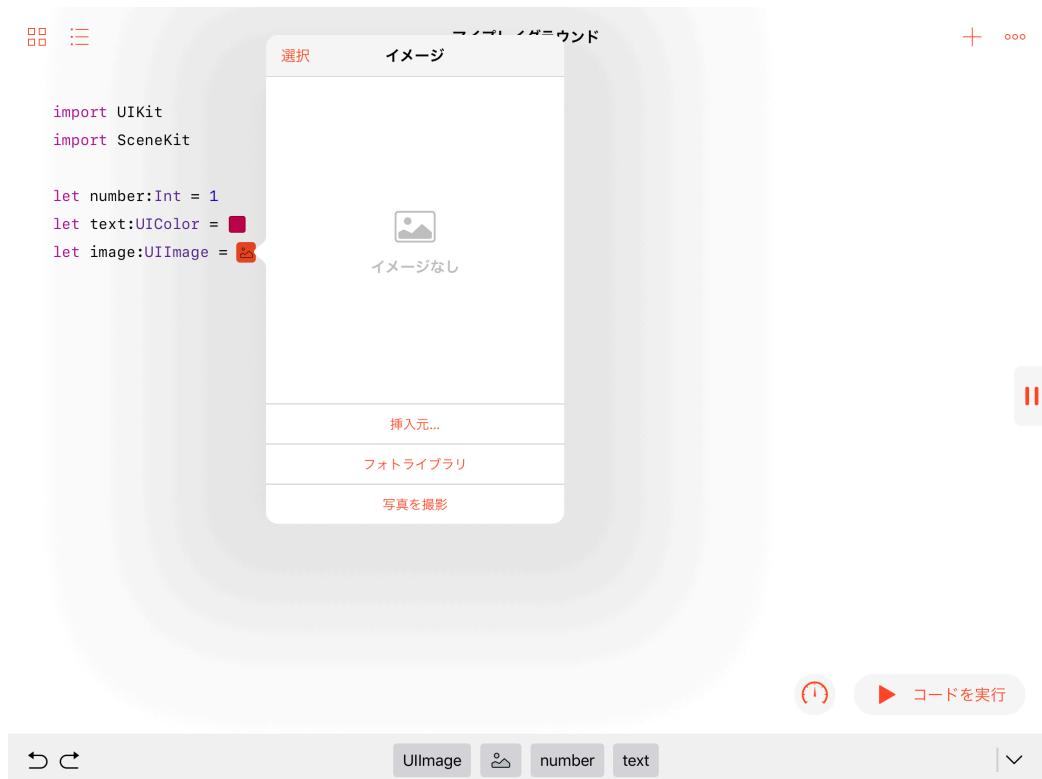
### 2.14 カラーリテラル

キーボード上のアクセサリビューに表示されるコード補完の角丸を選択するとカラーリテラルとなり、カラーパレットをから選択できます。



### 2.15 イメージリテラル

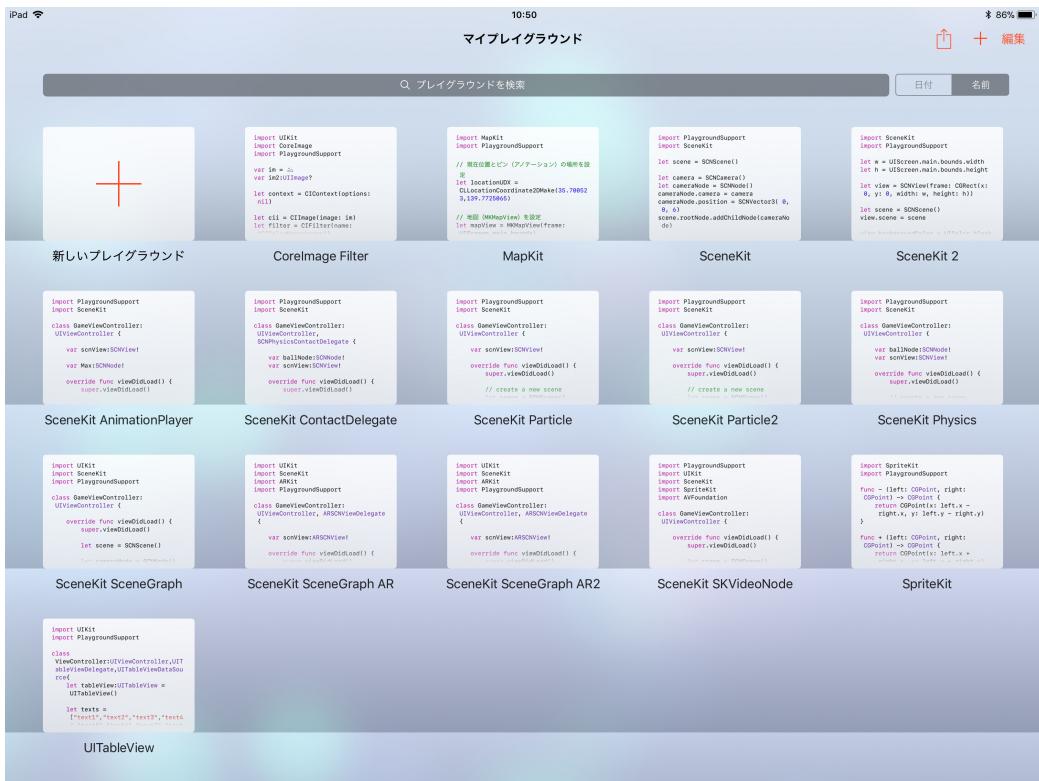
キーボード上のアクセサリビューに表示されるコード補完の写真のようなアイコンを選択するとイメージリテラルとなり、ファイルから選択できます。



## 2.16 Playground ファイルの検索とソート

マイプレイグラウンドで下にフリックすると検索窓と並び順を設定するボタンが出ます。並び順の初期値で状態である「日付」に設定していると更新したものが先頭にきますが、ファイルが多くなるとわかりづらくなるため「名前」での並び順をおすすめします。

## 第2章 Swift Playgrounds for iPad の使い方



# 第3章

## iOS アプリの概要

---

iPad の Swift Playgrounds を使用するということは、iOS のアプリを作成することと同様ですが、面倒な部分を省いて作成することができるので心配はいりません。

### 3.1 Swift Playgrounds for iPad での iOS アプリの仕組み

本来、Xcode 作成する iOS アプリでは、プロジェクトファイルや設定のファイルがありますが、Swift Playgrounds for iPad では必要ありません。正確には iPad 上の Swift Playgrounds では見ることができず細かい設定をしなくとも済みます。

その代わりに、アプリの申請や開発アカウントに紐づく機能など、いくつかの機能が Swift Playgrounds では使用することができません。

#### 3.1.1 Swift Playgrounds for iPad と Xcode で作成する際の違い

通常、大枠である UIWindow が見た目の設定とコントロールを行う UIViewController を呼び、UIViewController へ View (UIView / SceneKit の View) を設定を行います。

Swift Playgrounds の場合は UIWindow ではなく PlaygroundSupport フレームワークの liveView を使用します。

また、liveView は UIViewController ではなく View を渡しても表示することができます。

### 3.2 試しにアプリをつくる

地図を表示するアプリを作成します。

## 第3章 iOS アプリの概要

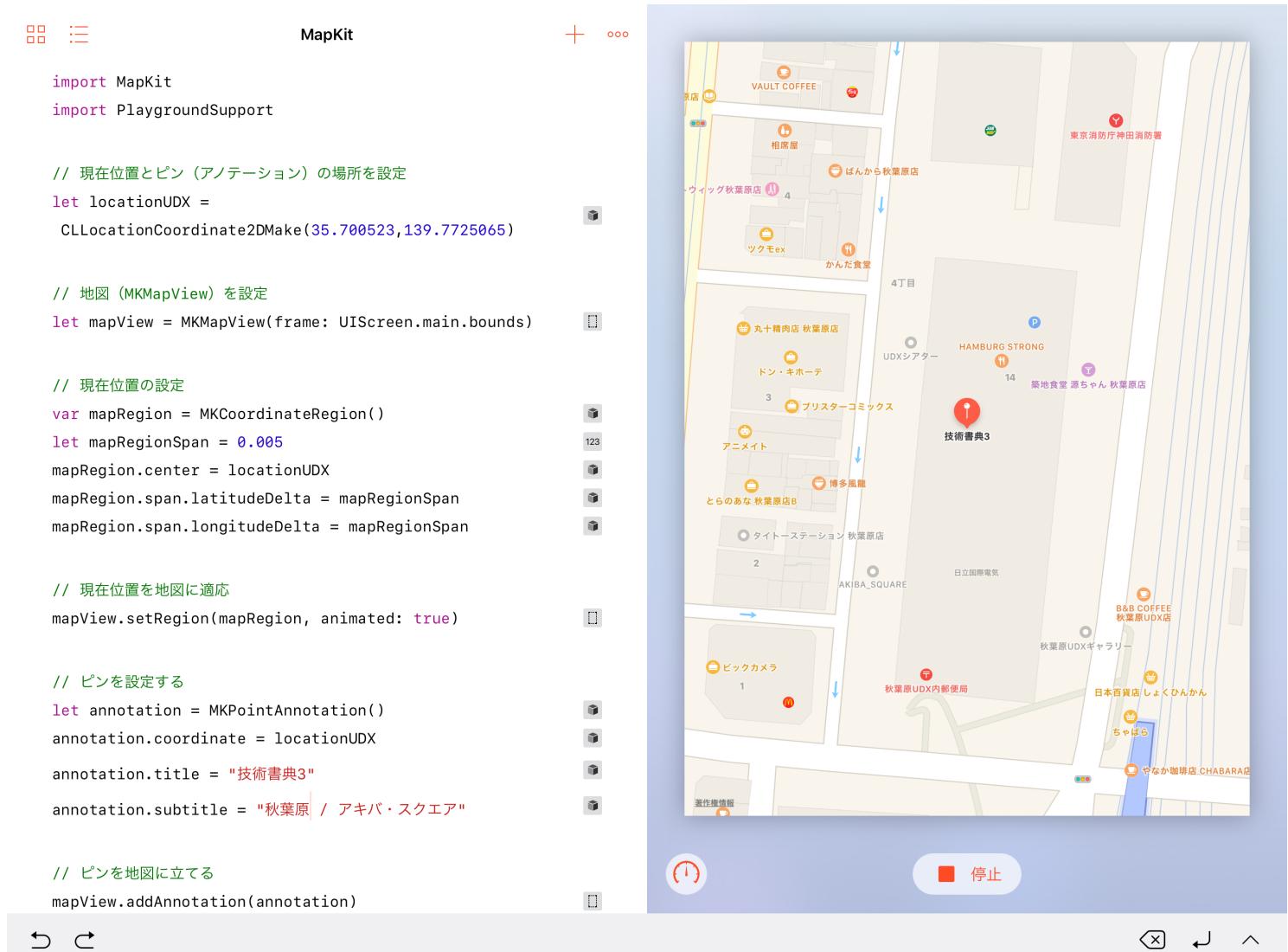


図 3.1 Map アプリ

以下のコードを Swift Playgrounds の空白のページへコピーし、「コードを実行」のボタンを押します。

```
import MapKit
import PlaygroundSupport

// 現在位置とピン（アノテーション）の場所を設定
let locationUDX = CLLocationCoordinate2DMake(35.700523, 139.7725065)

// 地図（MKMapView）を設定
let mapView = MKMapView(frame: UIScreen.main.bounds)
```

```
// 現在位置の設定
var mapRegion = MKCoordinateRegion()
let mapRegionSpan = 0.005
mapRegion.center = locationUDX
mapRegion.span.latitudeDelta = mapRegionSpan
mapRegion.span.longitudeDelta = mapRegionSpan

// 現在位置を地図に適応
mapView.setRegion(mapRegion, animated: true)

// ピンを設定する
let annotation = MKPointAnnotation()
annotation.coordinate = locationUDX
annotation.title = "技術書典 3"
annotation.subtitle = "秋葉原 UDX / アキバ・スクエア"

// ピンを地図に立てる
mapView.addAnnotation(annotation)

// Playground Live View に地図を適応して表示
PlaygroundPage.current.liveView = mapView
```

今回は SceneKit の説明なので、このコードの説明は割愛しますが、JavaScript で Google マップを表示するぐらいのコード量でアプリが作成できているのがわかると思います。

### 3.3 コードを軽く変更する

新しい場所の位置を設定し、地図が画面に表示される際に秋葉原駅を中心にします。

#### 3.3.1 変更するコード

let locationUDX の下に以下を追加して、

```
let location = CLLocationCoordinate2DMake(35.698353, 139.7731143)
```

mapRegion.center の locationUDX を location に変更する。

```
mapRegion.center = location
```

実行ボタンを押すと、今までではピンがマップの中心にありました。秋葉原駅が中心になっていると思います。

このようにアプリの作成や修正がわりと簡単で iPad でできました。

# 第4章

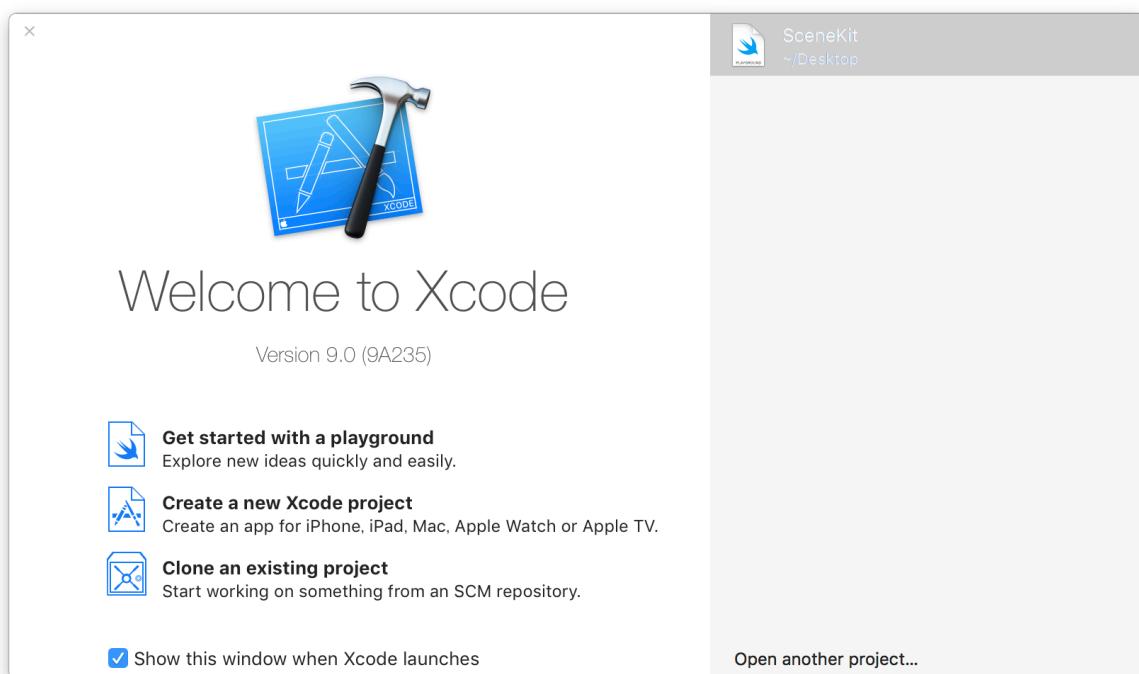
## Xcode 版 Playgrounds

基本的には iPad と使い方は同じですが、Xcode 版の Swift はビルド時に iOS シミュレーターが起動、実行されるため、iPad がお手元にある場合は iPad を使用した方が、ビルド、実行は速いです。

Xcode 9 がインストールされていない場合は、App Store から Xcode を検索しインストールしてください。

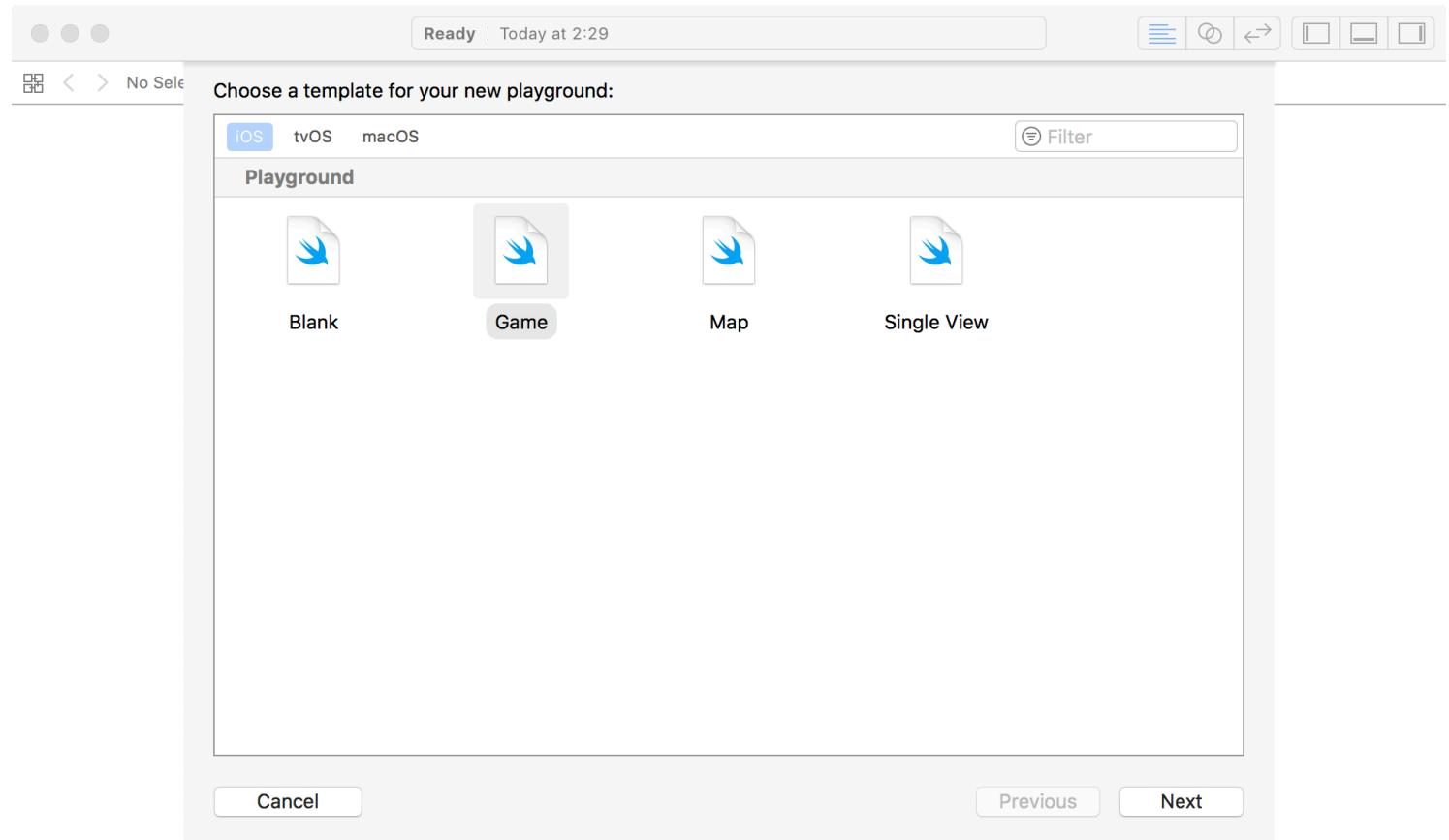
### 4.1 Xcode 版の Playgrounds で新規の Playground ファイル

Xcode 起動すると「Welcome to Xcode」が表示されるので、そこにある「Get Start with a playground」をクリックするか、Command + Shift + Option + N を押します。



## 第4章 Xcode 版 Playgrounds

テンプレート選択画面が表示されるので、上部タブ「iOS」を選択し、「Game」テンプレートを選択し、下の「Next」ボタンをクリックすると新規作成されます。



Xcode 版の Playgrounds は初期状態では自動ビルド実行されますが最初はシミュレーター起動に時間がかかりますので、しばらく待ちましょう。

Assistant Editor を Command + Option + Return キーで開き、そのまま放置していると Hello World と表示されるアプリが起動します。

## 第4章 Xcode 版 Playgrounds

```
1 //: A SpriteKit based Playground
2
3 import PlaygroundSupport
4 import SpriteKit
5
6 class GameScene: SKScene {
7
8     private var label : SKLabelNode!
9     private var spinnyNode : SKShapeNode!
10
11    override func didMove(to view: SKView) {
12        // Get label node from scene and store it for use later
13        label = childNode(withName: "//helloLabel") as? SKLabelNode
14        label.alpha = 0.0
15        let fadeInOut = SKAction.sequence([.fadeIn(withDuration: 2.0),
16                                           .fadeOut(withDuration: 2.0)])
17        label.run(.repeatForever(fadeInOut))
18
19        // Create shape node to use during mouse interaction
20        let w = (size.width + size.height) * 0.05
21
22        spinnyNode = SKShapeNode(rectOf: CGSize(width: w, height: w),
23                               cornerRadius: w * 0.3)
24        spinnyNode.lineWidth = 2.5
25
26        let fadeAndRemove = SKAction.sequence([.wait(forDuration: 0.5),
27                                               .fadeOut(withDuration: 0.5),
28                                               .removeFromParent()])
29        spinnyNode.run(.repeatForever(.rotate(byAngle: CGFloat(Double.pi),
30                                      duration: 1)))
31        spinnyNode.run(fadeAndRemove)
32
33    func touchDown(atPoint pos : CGPoint) {
34        guard let n = spinnyNode.copy() as? SKShapeNode else { return }
35
36        n.position = pos
37        n.strokeColor = SKColor.green
38        addChild(n)
39
40    func touchMoved(toPoint pos : CGPoint) {
41        guard let n = self.spinnyNode.copy() as? SKShapeNode else { return }
42
43        n.position = pos
44        n.strokeColor = SKColor.blue
45    }
46
47 }
```

アプリの領域をクリックすると、クリック位置からクリックが終わるまでに角丸の矩形がアニメーション付きで描画されます。

Game テンプレートは SpriteKit で書かれており、本書のサンプルコードに書き換えることで Mac 上でもサンプルを試すことができます。

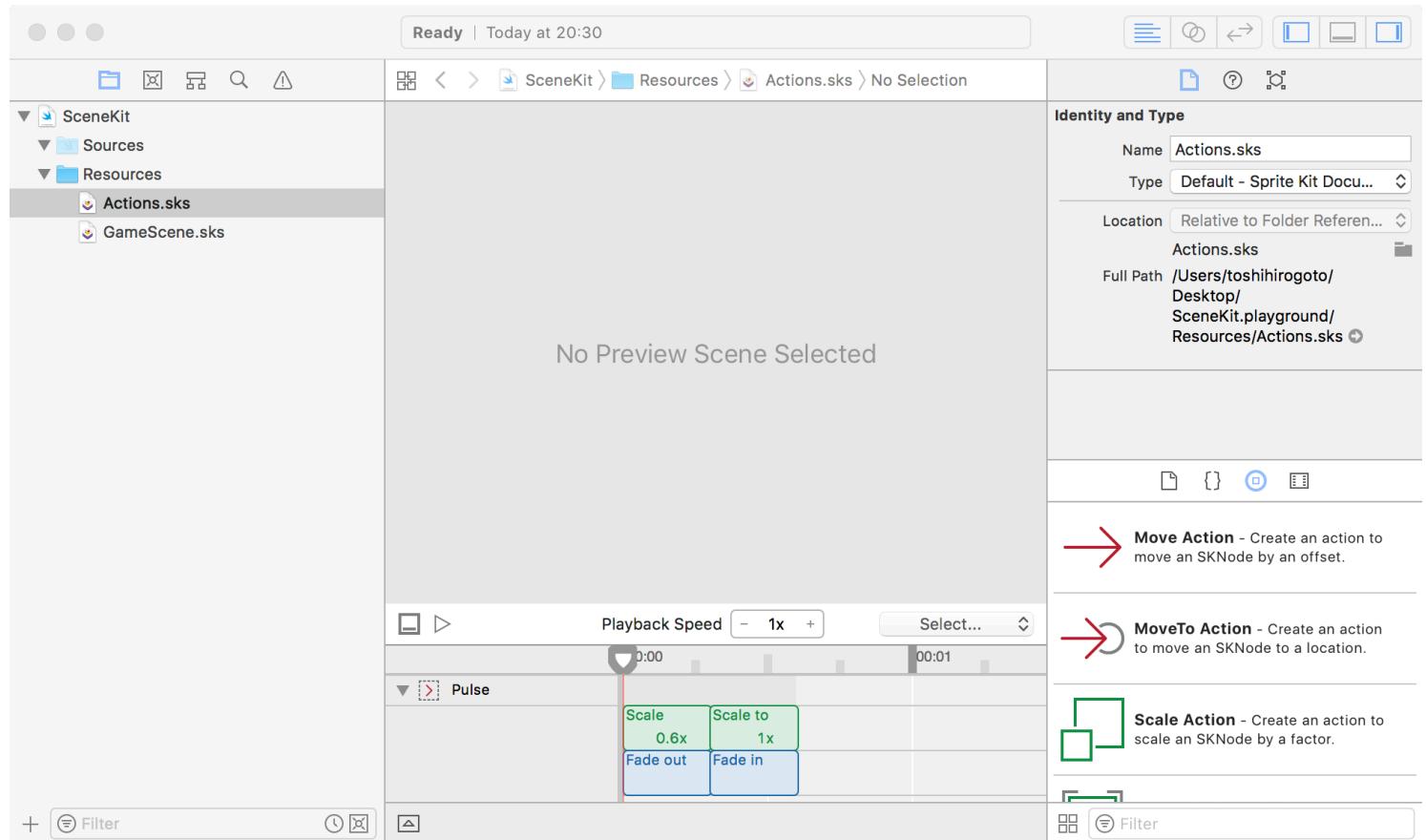
## 4.2 Xcode の Playgrounds で使用するリソースの保存方法

テクスチャ画像やジオメトリのデータ、音などの

通常の Xcode と同様に Command + 1 で Project Navigator を開きと「Resources」に使用するファイルをドラッグ&ドロップします。

Game テンプレートで作成する Playground ファイルでは Resources のフォルダに、Actions.sks と GameScene.sks という SpriteKit のシーンファイルが設定されています。

## 第4章 Xcode版Playgrounds



### 4.3 iPadで作成したPlaygroundファイルをXcodeのPlaygroundsで実行する

iCloud Driveでの同期を有効にしている場合、iPadのSwift Playgrounds側での保存するとMac側のiCloud Driveでも同期され、ダブルクリックで起動します。

iCloud Driveを使用していない場合は、AirDrop、メールなどでPlaygroundファイルをMacで取得し実行することができます。

また、ARKitなど実機でしか動作しない命令はMacで実行することはできません。

### 4.4 Xcode 版の Playgrounds のビルド、実行が遅い場合

遅い場合や Metal を使用している場合は macOS のアプリか、実機へビルドできる場合は iOS のアプリとして作成します。次の章で紹介します。

# 第5章

## Xcode で macOS アプリを作成する

---

Xcode の Playground で NSViewController を扱うのは面倒なので、第3章で作成したコードを修正し macOS のアプリでも動くようにします。

### 5.1 iOS から macOS に変更する際に、修正する命令

iOS から macOS プラットフォームを変更しても、そのままでは動かないため、一部使用している命令を変更します。

iOS	macOS
import UIKit	import Cocoa
UIColor	NSColor
UIImage(named:"image.png")	NSImage.Name(rawValue: "image.png")

主に変更するものは NSColor と NSImage ぐらいで、共にリテラルで記述すると全く同じコードで動きます。

macOS の画像の読み込みですが、 **NSImage(named:"image.png")** ではない点に注意してください。

### 5.2 Project の作成

Xcode 起動後 「Create New Xcode project」 を選択。macOS を選択し、Game テンプレートを選択し「Next」をクリック。

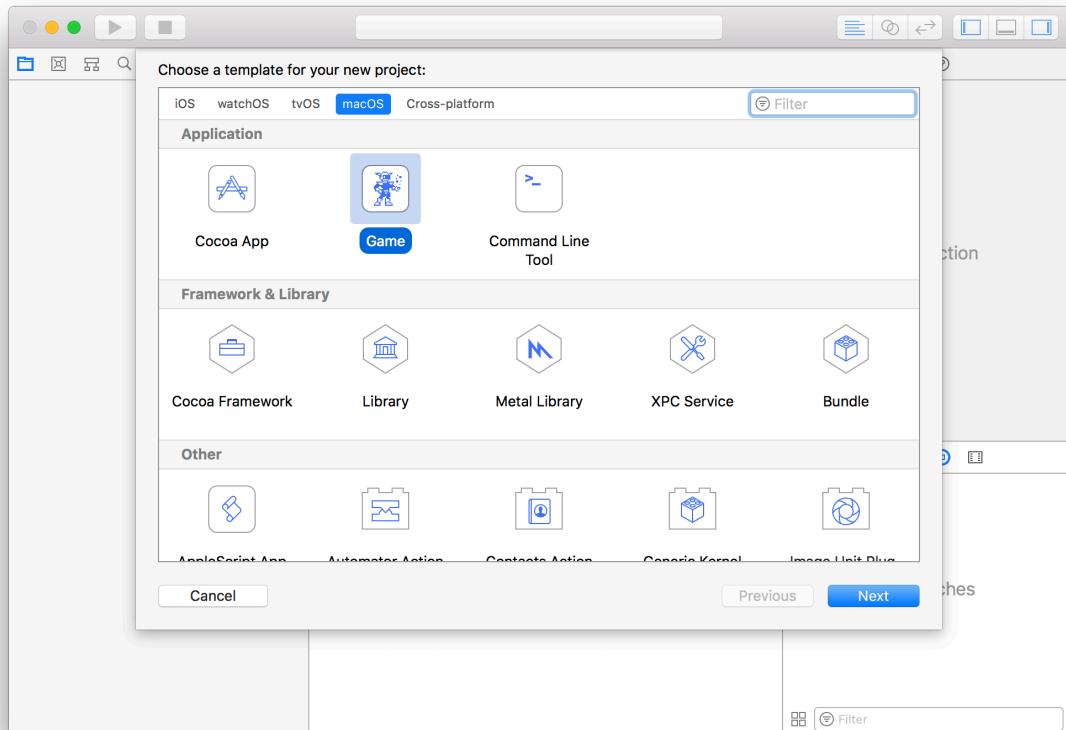


図 5.1 Game テンプレートを選択

ファイル名の設定し Game Technology を SceneKit を選択し「Next」をクリック。(今回は SceneKit を使用しないのですが後のもののために選んでいます)

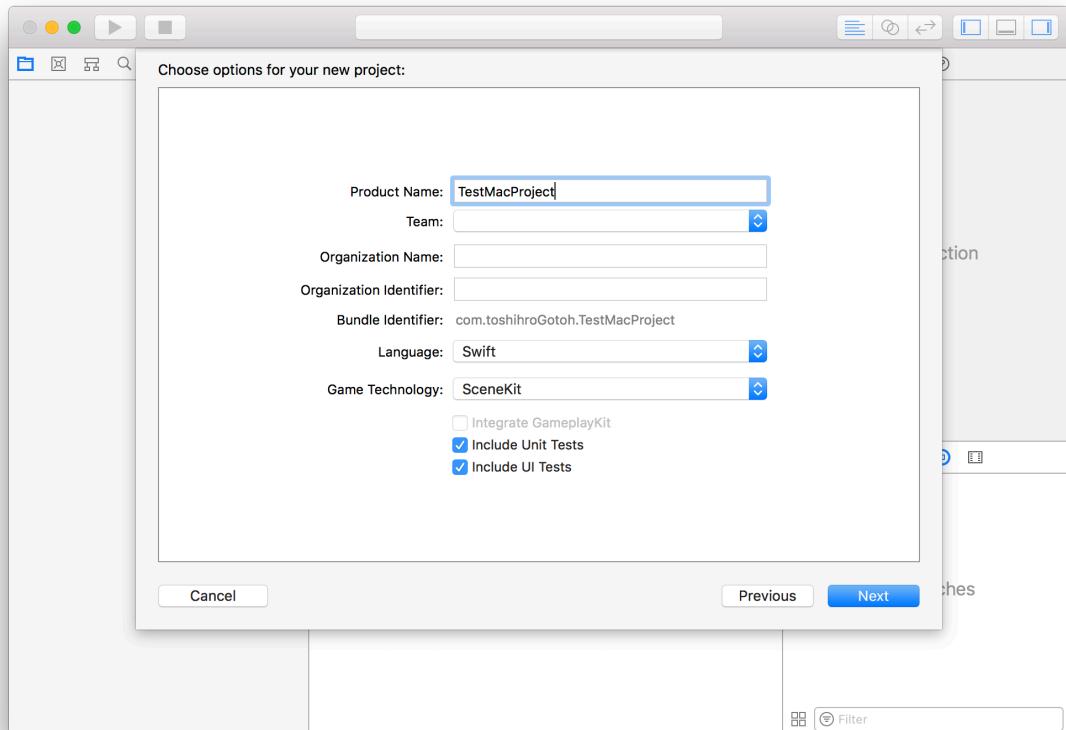


図 5.2 SceneKit を選択

プロジェクトが表示されます。

### 5.3 コードを書く

Project Navigator (Command + 0) の GameViewController.swift を開いて全て消し以下を記述します。

リスト 5.1: NSViewController

```
import MapKit

class GameViewController: NSViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

    }
}
```

viewDidLoad() の中の super.viewDidLoad() に第3章で作成したコードをコピーし、2箇所変更を行います。

- UIScreen.main.bounds > self.view.bounds
- PlaygroundPage.current ~ > self.view.addSubview(mapView)

リスト 5.2: GameViewController.swift

```
import Cocoa
import MapKit

class GameViewController: NSViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // 現在位置とピン（アノテーション）の場所を設定
        let locationUDX = CLLocationCoordinate2DMake(35.700523, 139.7725065)

        // 地図（MKMapView）を設定
        let mapView = MKMapView(frame: (self.view.bounds))

        // 現在位置の設定
        var mapRegion = MKCoordinateRegion()
        let mapRegionSpan = 0.005
        mapRegion.center = locationUDX
        mapRegion.span.latitudeDelta = mapRegionSpan
        mapRegion.span.longitudeDelta = mapRegionSpan

        // 現在位置を地図に適応
        mapView.setRegion(mapRegion, animated: true)

        // ピンを設定する
        let annotation = MKPointAnnotation()
        annotation.coordinate = locationUDX
        annotation.title = "技術書典 3"
        annotation.subtitle = "秋葉原 UDX / アキバ・スクエア"

        // ピンを地図に立てる
        mapView.addAnnotation(annotation)
    }
}
```

```
    self.view.addSubview(mapView)
}

}
```

## 5.4 実行結果

記述が完了したら、Command + R でビルドし、以下にアプリが表示されます。



図 5.3 実行結果

ウインドウサイズ変更時の命令がないため、ウインドウサイズ変えるとピンの吹き出しが出なくなりますが、今回は表示のサンプルなので大目にみてください。

# 第6章

## SceneKit の概要

---

### 6.1 SceneKit とは？

Apple が提供する 3DCG のフレームワークで、Unity や Unreal Engine などのゲームエンジンに近いものとなっています。

ゲームエンジン同様に手軽に 3DCG を表示することができ、SceneKit を大雑把に説明すると iOS SDK が提供しているゲームエンジンという感じものとなります。

### 6.2 対応プラットフォーム

iOS SDK のフレームワークですので Windows や Android には対応していませんが、Apple が提供している全ての OS で使用可能となっています。

- iOS (iPhone/iPad/iPod touch)
- watchOS
- tvOS
- macOS

### 6.3 SceneKit の中身と関連するフレームワーク

#### 6.3.1 SceneKit の中身

SceneKit の中身は Metal か OpenGL (ES) かのいずれかの GPU を扱うハードウェア寄りのライブラリと、3DCG で扱うジオメトリ、画像、その他オブジェクトを管理や操作を行う Model I/O で構成されています。

### 6.3.2 関連するフレームワーク

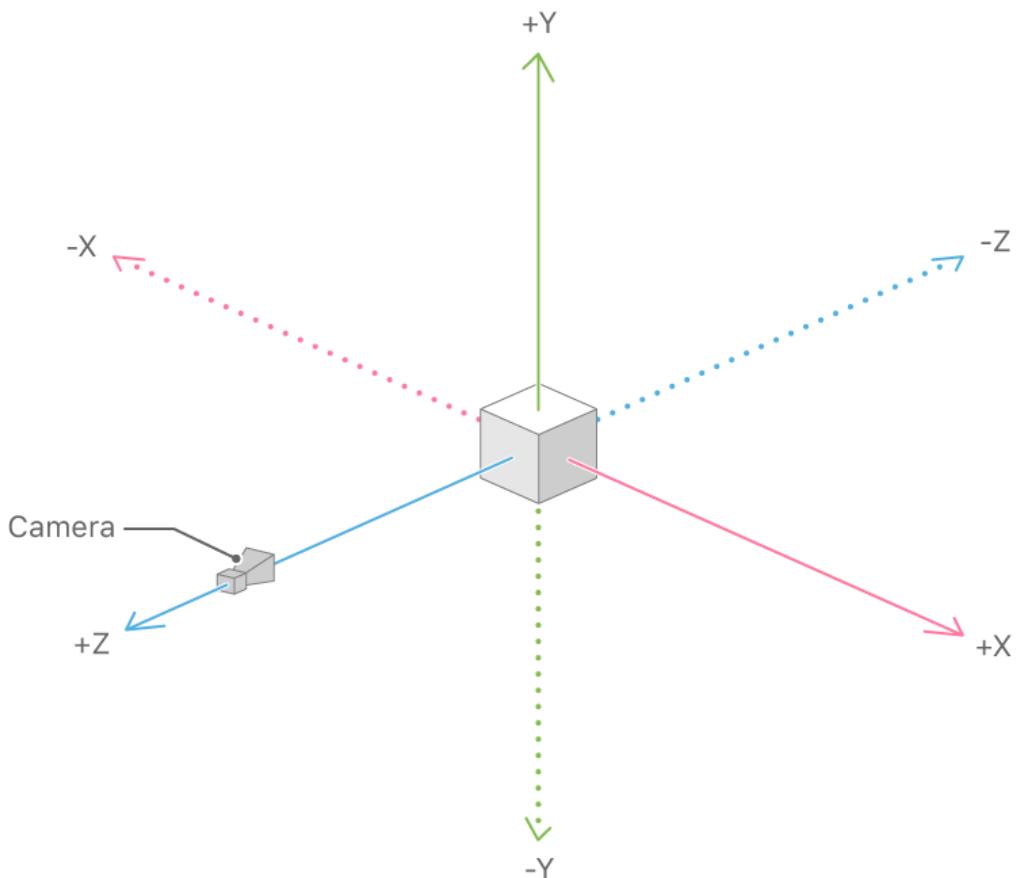
SceneKit は 3DCG を手軽に表示させることが主な仕事で、ゲーム制作をする補助するフレームワークとして以下ものが用意されています。

- ReplayKit - 画面の録画、ライブ配信を可能にする
- GameController - Bluetooth のゲームコントローラーを使用できるようにする
- GamePlayKit - ゲームロジック、オブジェクト追跡、コンポーネント化、乱数派生などの補助機能

## 6.4 SceneKit 座標系について

SceneKit は Y 軸が上を向く右手系の座標です。

そのため、奥行きが Z 軸で手前がプラスになっています。



画像はドキュメントの Organizing a Scene with Nodes から参照<sup>\*1</sup>

### 6.5 OpenGL (ES) と Metal

リアルタイムの3DCGを行うためのライブラリがいくつかあり、OpenGL(ES)とMetalはSceneKitの基盤となる部分で使用されており、以下は簡単な説明となります。

#### 6.5.1 OpenGL

シリコングラフィックスのワークステーションのOS、IRIXで使用されていたものが、オープンソースとなったもので、主なOSでほぼ全ての対応し、現在はKhronos Groupが仕様の策定を行なっています。

#### 6.5.2 OpenGL ES

モバイル版のOpenGLとなり、モバイルのGPUの制限のためOpenGLと比べると機能制限があり構文も異なります。新しいOpenGLではESに近い記述になりました。

#### 6.5.3 Metal

Appleが開発したグラフィックライブラリでSwiftやSpriteKit、SceneKitと一緒に発表されました。MicrosoftのDirectXの様に、ハードとOSに最適化された設計となっており、リリース時にはOpenGLより最大10倍速い謳っていました。WWDC 2017ではMetal2が発表され、Metalよりさらに最大10倍速い謳っています。

現状、SceneKitではMetalでしか動作しないものがかなり増えてきたため、OpenGLを選ぶ必要性が薄れています。また、Xcodeでプロジェクト新規作成でOpenGLのテンプレートが存在しないためApple的には積極的に使ってもらおうとはしていない様で、他のプラットフォームからの移植用として残されている雰囲気なのではと感じています。

---

<sup>\*1</sup> 参照元：[https://developer.apple.com/documentation/scenekit/organizing\\_a\\_scene\\_with\\_nodes](https://developer.apple.com/documentation/scenekit/organizing_a_scene_with_nodes)

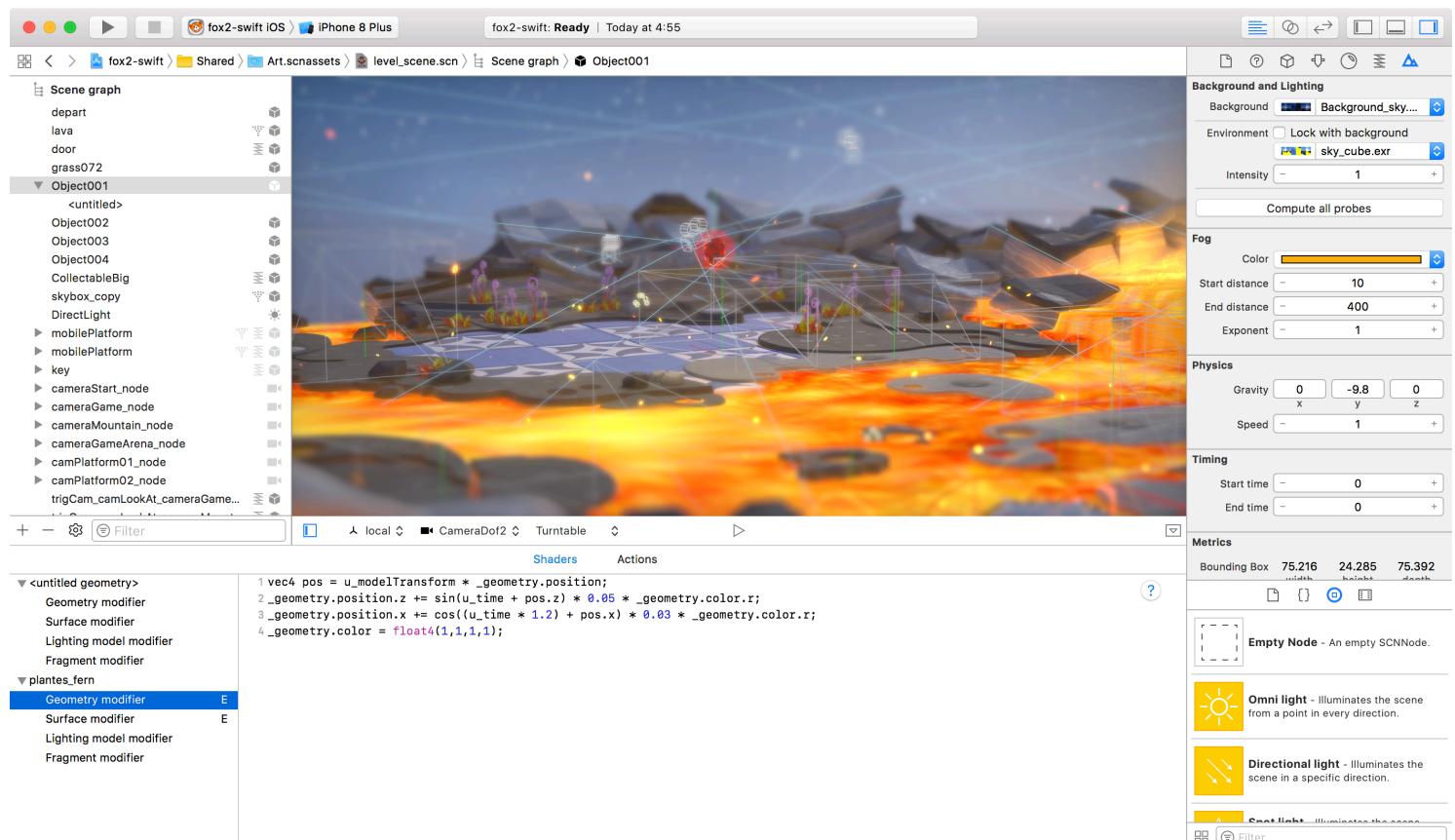
## 6.6 XcodeでのSceneKit

本書ではありませんがXcodeのSceneKitには以下の機能があります。

### 6.6.1 Scene Editor

Xcodeでの作業は基本的にはプログラムを作成することですのでテキスト入力する作業を中心になりますが、SceneKitでは3Dの空間でジオメトリを扱うため、視覚的に操作を行うこととなります。

普段はSource Editorでコードを扱いますが、シーンファイルファイル(.scn)、パーティクルシステムファイル(.scnp)、3DCGのオブジェクトファイルに関してはScene Editorという、Unityなどのゲームエンジンの操作画面のようUIに変更されます。

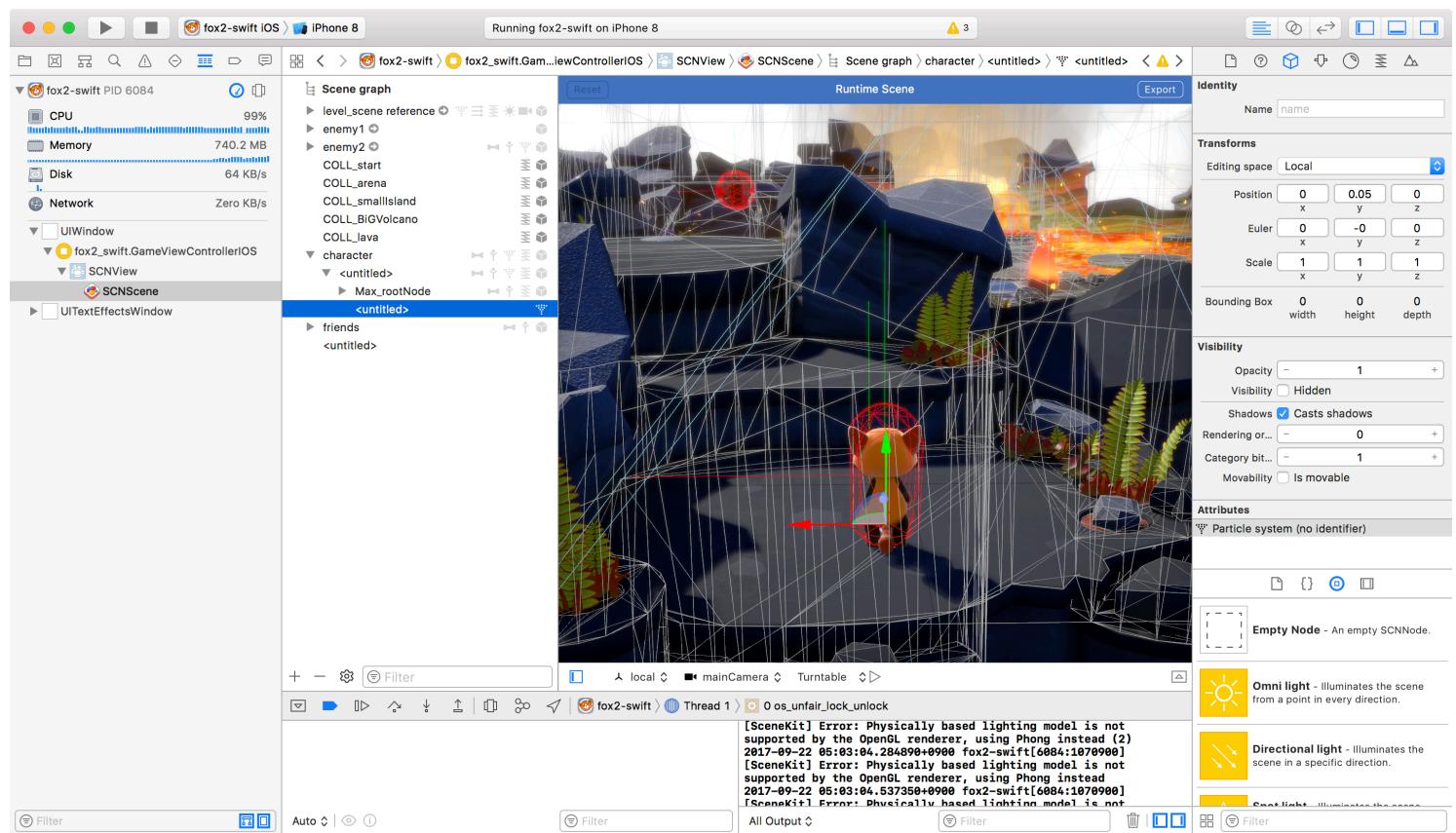


Xcode 9では設定項目が変更され、ShaderのスニペットがScene Editorで変更したり表示したりできます。

## 6.6.2 Debug

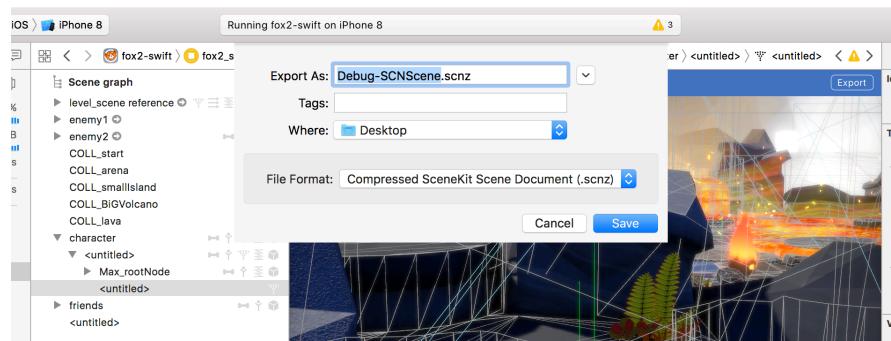
Xcode 9 からデバッグ実行時に Debug Navigator での View Hierarchy の階層にシーンを選択できる部分があり、そのまま、Scene Graph の構造を確認したり、各種インスペクタから状態を見たりすることができます。

Hierarchy から、シーンを選択すると画面が止まるため、動的には変更できず、現状を確認するためのものです。



また、上部の Export をクリックすると、dae、scn、scnz のいずれかに保存することができるようになりました。

## 第6章 SceneKitの概要



Xcode ではないのですが Instruments で SceneKit 実行状態を調べることができるようになりました。

# 第7章

## ほんの少しの数学の話と 3DCG のプログラム使用する技術

---

数学面ではラジアンと行列、プログラム面では法線と UV マップについて紹介します。小難しく書いていますが、本書ではラジアンぐらいしか直接使うものはないです。

### 7.1 ラジアン (rad)

定義的には半径と同じ長さの弧を中心とした角度が 1 ラジアンとなり、1 ラジアンを度数法にすると 57.29578 度になります。

わかりづらいので、簡単に説明すると  $1 \pi \text{ rad}$  が 180 度で、 $2 \pi \text{ rad}$  が 360 度になります。数学的にこの方が計算しやすいという話でこういうものだと思ってください。

なぜ、ここでラジアンの説明をしているかというと、SceneKit でオブジェクトを回転する際に使用しているためです。

一応、度数法に変換する場合は、 $\text{角度} \times (180/\pi)$  になります。例えば、90 度をラジアンに変化する場合は  $90 \times (180/\pi)$  となり、 $90 \times (180/\pi) = \pi/2 = 0.5 \pi$  となり、 $1 \pi \text{ rad}$  が 180 度なので、 $0.5 \pi$  は 90 度になります。

### 7.2 行列

SceneKit に限らず 3DCG で移動、回転、拡大縮小など、ジオメトリの操作は内部的に、 $4 \times 4$  の 16 個の値を持つ行列を使います。利点としては行列を使用すると移動、回転、拡大縮小を 1 回の計算で同時にを行うことができます。

SceneKit で便利な命令群が用意されているので必要はないのですが、行列を使用すると以下のようになり、xyz の座標のものを Y 軸の方向に  $\theta$  分回転すると x'y'z' を返す場合の式

です。

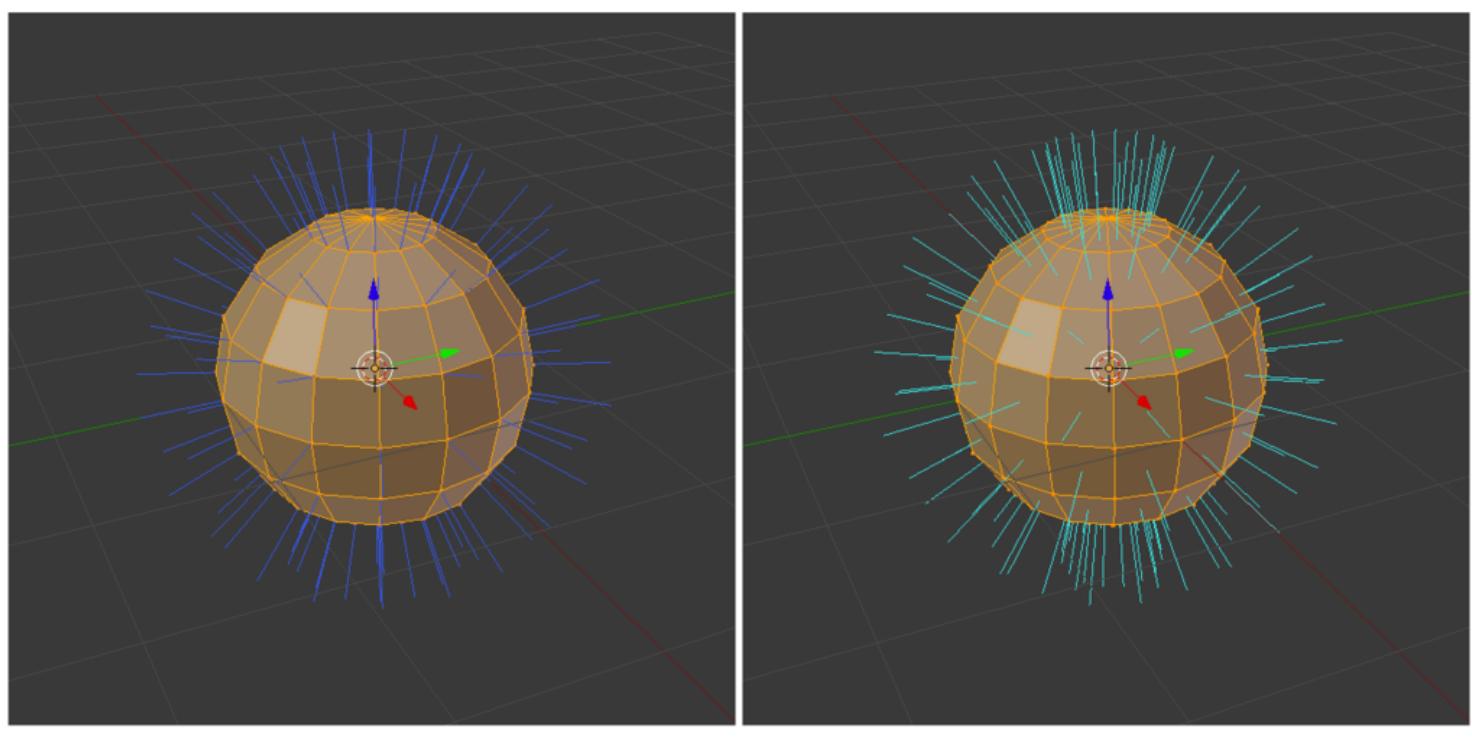
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

### 7.3 法線

法線は主な役目はジオメトリの頂点や面がどの方向を向いているかの情報です。

例えば、カメラやライトの光に対して表を見ているか裏を見ているか。カメラから見て裏にあるものは、ものに隠れるので描画しなくてもよいですし、ライトから見て裏にある部分は陰を落とします。

物理アニメーションやジオメトリの変形の際も使用され、マテリアルでは、最終的に映像を撮るカメラが、ライトとジオメトリとの関係を考えるために使用されます。



頂点の法線

面の法線

図 7.1 法線

## 7.4 UV マップ (Texture Coordinates)

ジオメトリにテクスチャを貼るための 2 次元座標です。テクスチャは 2 次元ですが、ジオメトリは 3 次元座標を持っているため、それをつけ合わせるための座標となります。

基本的には Maya など 3DCG の DCC (Digital Contents Creation) ツールを扱う方が設定します

こちらはカスタムジオメトリを作成の際に再度説明します。

# 第8章

## SceneKit の要であるシーディングラフ

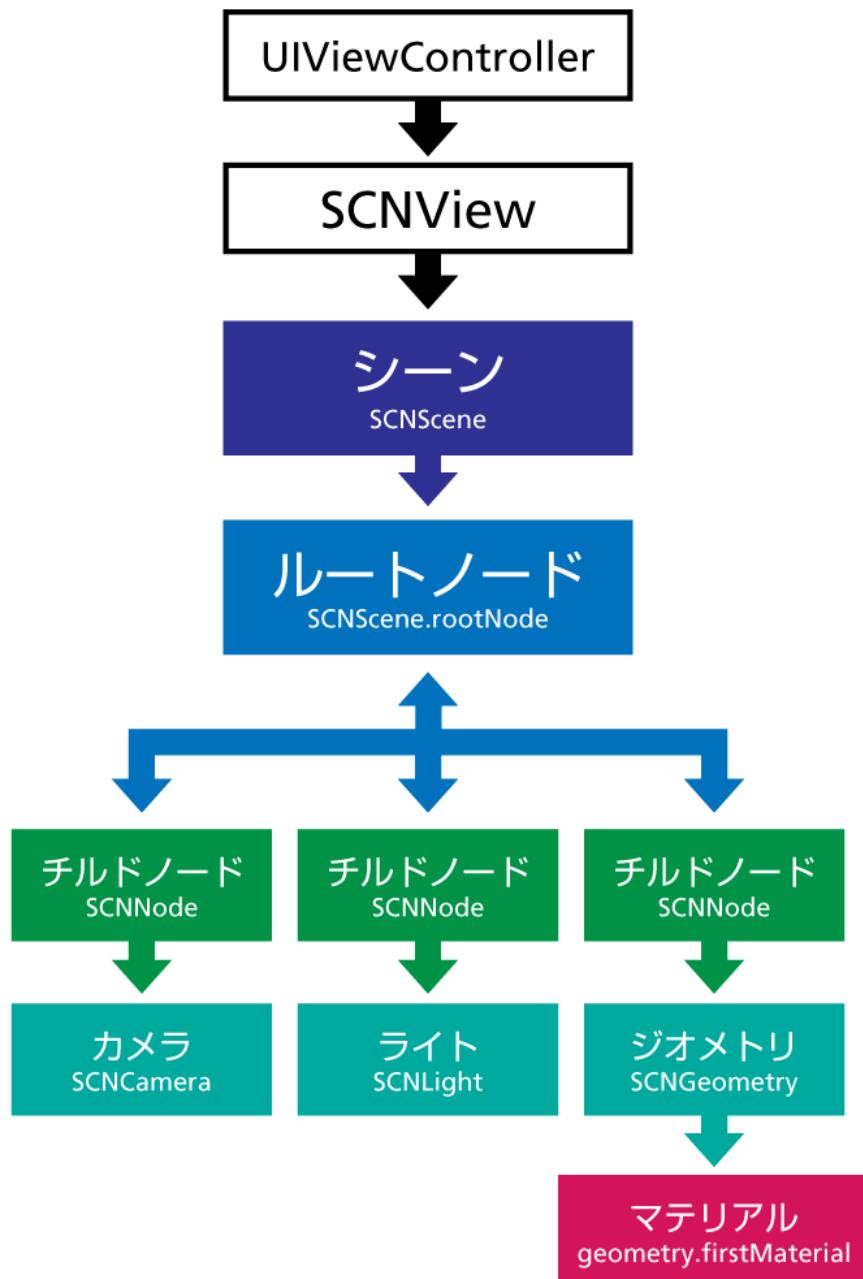
---

球を表示する SceneKit サンプルを作成しましたが、こちらもシーディングラフという仕組みの上で構成されています。この章ではシーディングラフについて見ていきましょう。

### 8.1 シーディングラフの構造

SceneKit はシーディングラフというツリー状の構造で画面構成をつくり 3DCG を表示しており、iOS の場合 UIViewController の View に SceneKit が用意している SCNView を適応します。

シーディングラフでは SCNScene が全体のシーン設定し、起点となるルートノードがぶら下がり、ルートノードから複数のチルドノードを設定する事ができて、各ノードにカメラやライトやオブジェクトを設定できます。



上の図では、チルドノードにカメラとライトとジオメトリしか書かれていませんが、パーティクルなど他の要素を追加することができます。また、チルドノードにチルドノードを追加し、さらに階層を深くすることもできます。

## 8.2 シーディングラフでの注意事項

Apple のドキュメントでは SceneKit は 60fps で動作することが望ましいと書かれています。シーディングラフの階層が深くなった場合、上の階層が変更されると画面描画の際、1秒間に

60 回調べる必要があるのでできるだけ簡素しましょう。iPad の Swift Playgrounds の場合、負荷が大きくなると実行中に処理ができなくなりエラーで止まる場合があります。

# 第9章

## Swift Playgrounds で階層化したノードの SceneKit を試す

SceneKit のシーディングラフがどのように使用されるか、サンプルの作成し試してみます。

```
SceneKit SceneGraph + ○○○  
import UIKit  
import SceneKit  
import PlaygroundSupport  
  
class GameViewController: UIViewController {  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        let scene = SCNScene()  
  
        let cameraNode = SCNNode()  
        cameraNode.camera = SCNCamera()  
        cameraNode.position =  
            SCNVector3(x: 0, y: 0, z: 0)  
  
        scene.rootNode.addChildNode(cameraNode)  
  
        let lightNode = SCNNode()  
        lightNode.light = SCNLight()  
        lightNode.light!.type = .omni  
        lightNode.position = SCNVector3(x:  
            0, y: 0, z: 0)
```

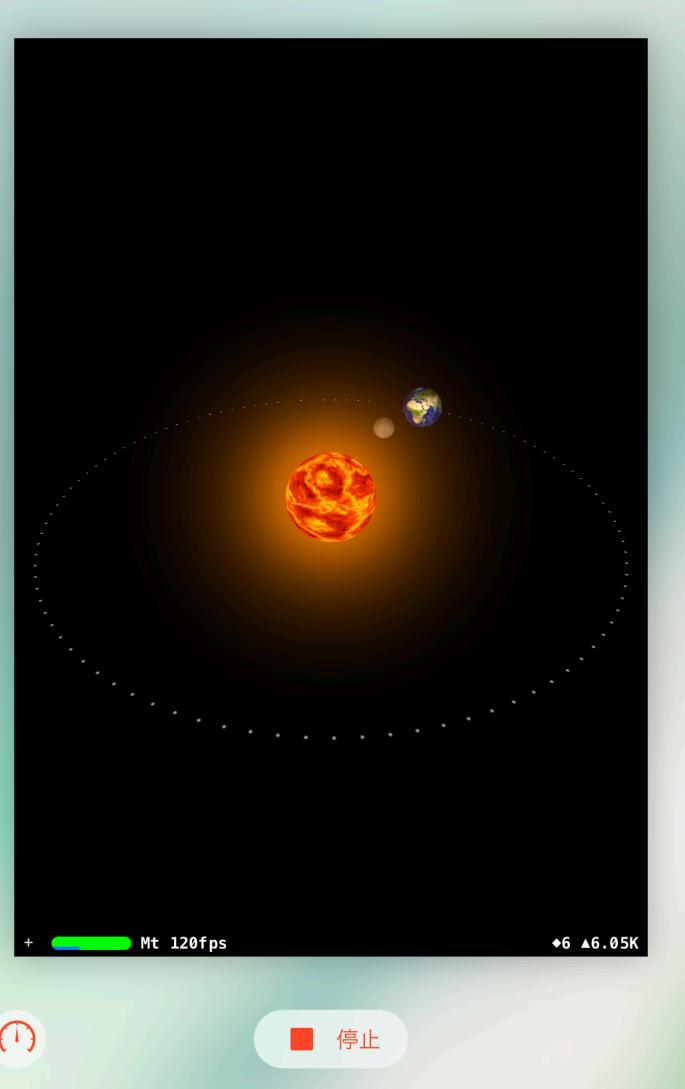


図 9.1 シーディングラフのサンプル

## 9.1 コードを書く前に

テクスチャを使用しているため、サンプルコードの Resources > SceneGraph のフォルダから全ての画像を設定します。iCloud Drive などに該当ファイルを保存してください。保存が完了したら空白のファイルの右上「+」から画像を Playground に追加します。

## 9.2 アプリの初期設定をする

はじめに LiveView に SCNView を表示させるため、以下を実行してみましょう。画面下にデバッグ関連の表示がされるかと思われます。

前回は View を PlaygroundPage.current.liveView 渡していましたが、今回は ViewController を使用しているため記述が長くなっています。

リスト 9.1: 初期設定

```
import UIKit
import SceneKit
import PlaygroundSupport

class GameViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // SCNScene の設定
        let scene = SCNScene()

        // カメラ設定
        let cameraNode = SCNNNode()
        cameraNode.camera = SCNCamera()
        cameraNode.position = SCNVector3(x: 0, y: 0, z: 0)
        scene.rootNode.addChildNode(cameraNode)

        // ここにコードを書く

        // SCNView の設定
        let scnView = SCNView()
        self.view.addSubview(scnView)
```

```
scnView.translatesAutoresizingMaskIntoConstraints = false

self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "V:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "H:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))


scnView.scene = scene
scnView.allowsCameraControl = true
scnView.showsStatistics = true
scnView.backgroundColor = UIColor.black
}

override var shouldAutorotate: Bool {
    return true
}

override var prefersStatusBarHidden: Bool {
    return true
}

override var supportedInterfaceOrientations: UIInterfaceOrientationMask {
    if UIDevice.current.userInterfaceIdiom == .phone {
        return .allButUpsideDown
    } else {
        return .all
    }
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Release any cached data, images, etc that aren't in use.
}

}

PlaygroundPage.current.liveView = GameViewController()
```

処置が終わると黒い画面が表示されます。

scene.rootNode.addChildNode(cameraNode) の部分でカメラがルートノードに設定されており、何も置かれていないため黒い空間になっています。

### 9.3 ライト、ノードに紐付いた球を置く

ライトとノードに紐付いた球を置いてみます。

#### 9.3.1 シーンのルートにライトを設定する

カメラの下に以下を記述します。

リスト 9.2: ライトを設定する

```
let lightNode = SCNNNode()
lightNode.light = SCNLight()
lightNode.light!.type = .omni
lightNode.position = SCNVector3(x: 0, y: 0, z: 0)
scene.rootNode.addChildNode(lightNode)

let ambientLightNode = SCNNNode()
ambientLightNode.light = SCNLight()
ambientLightNode.light!.type = .ambient
ambientLightNode.light!.color = UIColor.darkGray
scene.rootNode.addChildNode(ambientLightNode)
```

現状では、ジオメトリがないので変化がありません。

#### 9.3.2 シーンのルートにノードに紐付いた球を設置する

ライトの下に全ての起点となる空のノードを作成します。次章の都合上、今回はカメラがシーンの原点を取り、このノードが離れた場所に置かれます。

今回のように起点となるノードを使用し全体を動かす必要がなければ、ルートノードに直接ノードを並べても構いません。

リスト 9.3: 起点となる空のノードを作成

```
let baseNode = SCNNNode()
baseNode.position = SCNVector3(x: 0, y: 0, z: -30)
scene.rootNode.addChildNode(baseNode)
```

先ほどの baseNode のチルドノードとして球を設定します。

リスト 9.4: チルドノードとして球を設定

```
let sunNode = SCNNNode()
let sun = SCNSphere(radius: 2.5)
sun.firstMaterial?.diffuse.contents = UIImage(named:"sun.jpg")
sunNode.geometry = sun
sunNode.position = SCNVector3(0, 0, 0);

baseNode.addChildNode(sunNode)
```

## 9.4 球のチルドノードに球を配置する

先ほど 追加した球に対して、さらにノードを追加し、そのノードのチルドノードに球を配置します。中心から 15m 離れた位置に配置されます。

リスト 9.5: チルドノードに球を配置

```
let earthRotationNode = SCNNNode()
sunNode.addChildNode(earthRotationNode)

let earthGroupNode = SCNNNode()
earthGroupNode.position = SCNVector3(15, 0, 0)
earthRotationNode.addChildNode(earthGroupNode)

let earthNode = SCNNNode()
let earth = SCNSphere(radius: 1.5)
earth.firstMaterial?.diffuse.contents = UIImage(named:"earth-diffuse-mini.jpg")
earthNode.geometry = earth
earthNode.position = SCNVector3(0, 0, 0)
earthGroupNode.addChildNode(earthNode)
```

### 9.5 さらにもう一つ球を設定する

先ほど追加した球に対して、さらにチルドノードを追加しもう1つ球を設定します。

リスト 9.6: さらにチルドノードに球を配置

```
let moonRotationNode = SCNNode()
earthGroupNode.addChildNode(moonRotationNode)

let moonNode = SCNNode()
let moon = SCNSphere(radius: 0.75)
moon.firstMaterial?.diffuse.contents = UIImage(named:"moon.jpg")
moonNode.geometry = moon
moonNode.position = SCNVector3Make(5, 0, 0);
moonRotationNode.addChildNode(moonNode)
```

そして、軌道を表示するために平面の画像を起点のノードである `baseNode` に追加します。

リスト 9.7: 軌道の画像を配置

```
let earthOrbitNode = SCNNode()
earthOrbitNode.opacity = 0.6;
earthOrbitNode.rotation = SCNVector4(1, 0, 0, CGFloat(Float.pi * -0.5));

let earthOrbit = SCNPlane(width: 31, height: 31)
earthOrbit.firstMaterial?.diffuse.contents = UIImage(named:"orbit.png")
earthOrbit.firstMaterial?.lightingModel = .constant
earthOrbit.firstMaterial?.isDoubleSided = true

earthOrbitNode.geometry = earthOrbit

baseNode.addChildNode(earthOrbitNode)
```

### 9.6 太陽に輝きつける

太陽の輝きを画像として設置します。半透明にし、`SCNBillboardConstraint` を使用し画像が必ずカメラを向くようにしています。

### リスト 9.8: 太陽に輝きつける

```
let sunHalo = SCNPlane(width: 30, height: 30)
sunHalo.firstMaterial?.diffuse.contents = UIImage(named: "sun-halo.png")
sunHalo.firstMaterial?.emission.contents = UIImage(named: "sun-halo.png")
sunHalo.firstMaterial?.lightingModel = .constant
sunHalo.firstMaterial?.writesToDepthBuffer = false

let sunHaloNode = SCNNNode()
sunHaloNode.opacity = 0.4;
sunHaloNode.constraints = [SCNBillboardConstraint()]
sunHaloNode.geometry = sunHalo

sunNode.addChildNode(sunHaloNode)
```

## 9.7 アニメーションをつける

太陽を中心に地球と月が回るようにします。

### 9.7.1 アニメーションさせるノードの構造

太陽に地球の空のノードが結びついており、その空のノードに地球のジオメトリが結びつき、地球の空のノードに月のノードが結びついており、その空のノードに月のジオメトリが結びついております。

### 9.7.2 アニメーションの流れ

- 地球の空のノードを回転させると太陽を中心に回転します。
- 地球のノードを回転させると地球が回転します。
- 月の空のノードを回転させると地球を中心に回転します。
- 月のノードを回転させると月が回転します。

### 9.7.3 コード

Action を使用し、repeatForever で無限再生、設定している秒数 (duration) すべて、Y 軸 360 度回転させています。

リスト 9.9: 各回転

```
// 地球の公転
earthRotationNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFLOAT(Float.pi * 2), z: 0, duration: 10)
    )
)

// 地球の自転
earthNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFLOAT(Float.pi * 2), z: 0, duration: 1)
    )
)

// 月の公転
moonRotationNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFLOAT(Float.pi * 2), z: 0, duration: 1.5)
    )
)

// 月の自転
moonNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFLOAT(Float.pi * 2), z: 0, duration: 1.5)
    )
)
```

### 9.7.4 太陽のテクスチャを調整しアニメーションさせる

マテリアルでのテクスチャを適応を multiply にも行い、テクスチャのリピートを設定。Core Animation を使用して、テクスチャを半分の大きさにしながら少しづつ移動を行うアニ

メーションをしています。

リスト 9.10: 太陽のテクスチャアニメーション

```
sun.firstMaterial?.multiply.contents = UIImage(named:"sun.jpg")
sun.firstMaterial?.multiply.intensity = 0.5;
sun.firstMaterial?.lightingModel = .constant

sun.firstMaterial?.diffuse.wrapS = .repeat
sun.firstMaterial?.diffuse.wrapT = .repeat
sun.firstMaterial?.multiply.wrapS = .repeat
sun.firstMaterial?.multiply.wrapT = .repeat

let sunAnimation = CABasicAnimation(keyPath: "contentsTransform")
sunAnimation.duration = 10
sunAnimation.fromValue = NSValue.init(caTransform3D:
CATransform3DConcat(CATransform3DMakeTranslation(0, 0, 0),
CATransform3DMakeScale(3, 3, 3)))
sunAnimation.toValue = NSValue.init(caTransform3D:
CATransform3DConcat(CATransform3DMakeTranslation(1, 1, 0),
CATransform3DMakeScale(3, 3, 3)))
sunAnimation.repeatCount = .infinity

sun.firstMaterial?.diffuse.addAnimation(sunAnimation, forKey: nil)
sun.firstMaterial?.multiply.addAnimation(sunAnimation, forKey: nil)
```

## 9.8 Xcode の iOS アプリとして実装する

Swift Playgrounds で作成したコードをコピーし 2箇所修正するだけです。

### 9.8.1 プロジェクト作成

macOS と同様に、Xcode 起動後「Create New Xcode project」を選択。上のタブで iOS を選択し、Game テンプレートを選択し「Next」をクリック。

ファイル名の設定し Game Technology を SceneKit を選択し「Next」をクリックします。

### 9.8.2 コード編集

Project Navigator (Command + 0) の GameViewController.swift を開き、全て消し今回のコードをペーストします。

3 行目の import PlaygroundSupport と、最後の PlaygroundPage.current.liveView = GameViewController() を削除すると完成です。

Command + R でビルドすると、シミュレーターもしくは実機で動作すると思われます。

# 第10章

## SceneKit のシーンを ARKit に移植する

---

ARKit は現実の空間に 2D や 3DCG の仮想的なオブジェクト配置した仮想空間を合成して映像をつくりだします。他の機能では現実空間の平面を認識が可能で、iOS デバイスのタッチパネル、加速度センサー、Bluetooth で接続するゲームパットなどを使用して仮想空間を操作することもできます。

ARKit では SpriteKit や SceneKit がラップされているため、わりと簡単に実装が行えます。今回は平面の認識はせず空間に置くだけなので、初期設定後、前章でつくったコードを配置し一部修正するだけです。

アプリ起動時に端末が SceneKit のシーンが原点を取り、見づらいので前章では `baseNode` を若干奥に移動させています。

カメラの使用許可を促すアラートが出ますので「はい」を押しましょう。

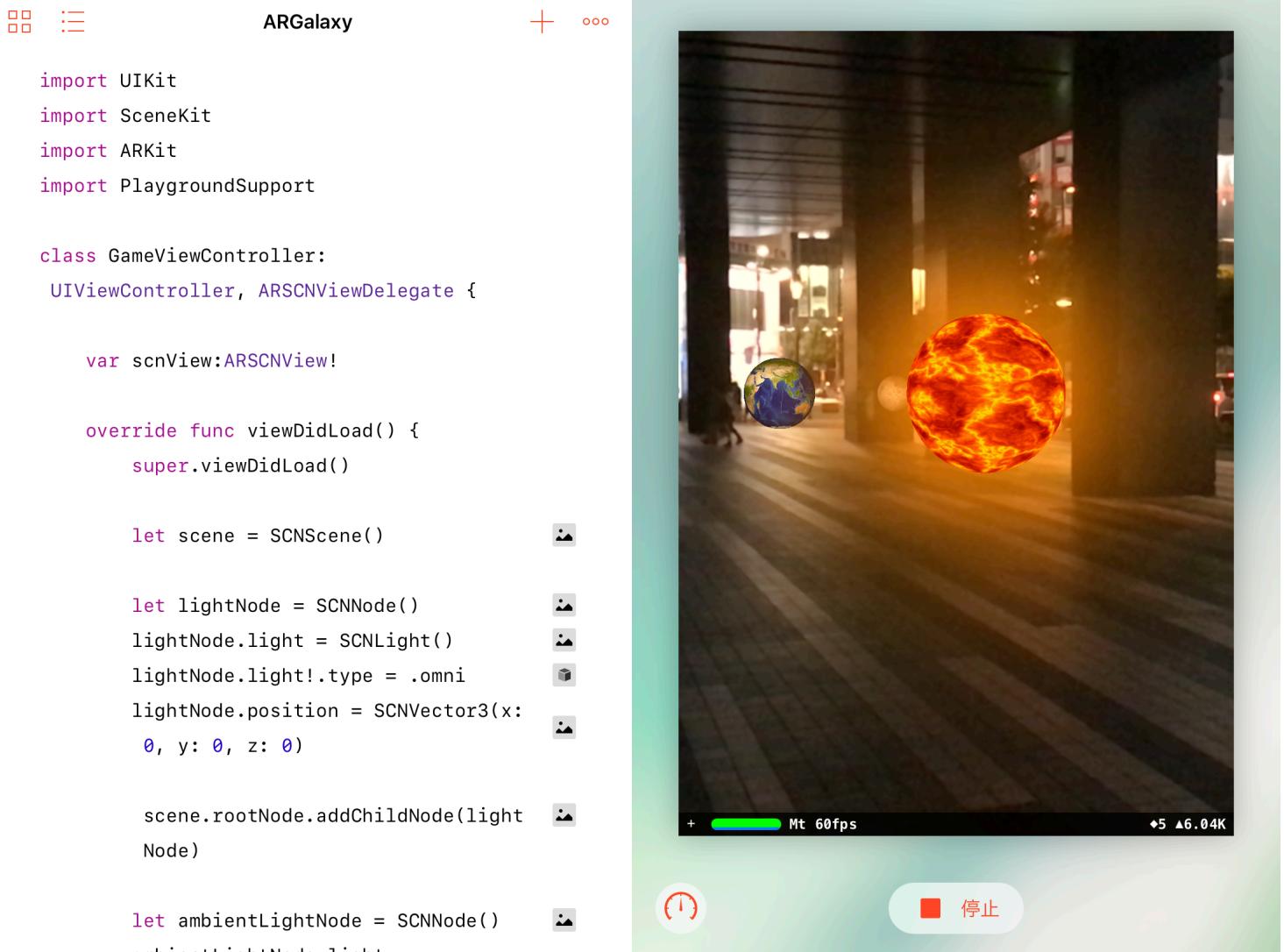


図 10.1 ARKit のサンプル

## 10.1 コードを編集する

### 10.1.1 初期設定

まずは初期設定から。「import ARKit」や ViewComntroller でコードが増えています。

また、一番最後の needsIndefiniteExecution を true にすることでカメラの使用許可を促すアラートが出るように設定しています。

リスト 10.1: 初期設定

```
import UIKit
import SceneKit
import ARKit
import PlaygroundSupport

class GameViewController: UIViewController, ARSCNViewDelegate {

    var scnView:ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        let configuration = ARWorldTrackingConfiguration()

        scnView.session.run(configuration)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        scnView.session.pause()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    func session(_ session: ARSession, didFailWithError error: Error) {}
    func sessionWasInterrupted(_ session: ARSession) {}
    func sessionInterruptionEnded(_ session: ARSession) {}

}

PlaygroundPage.current.liveView = GameViewController()
PlaygroundPage.current.needsIndefiniteExecution = true
```

## 10.1.2 前章コードの追加

前章の `viewDidLoad()` の中身をこちらのコードにペーストし、「`let scnView = SCNView()`」を「`scnView = ARSCNView()`」に変更してください。

リスト 10.2: 追加したコード

```
import UIKit
import SceneKit
import ARKit
import PlaygroundSupport

class GameViewController: UIViewController, ARSCNViewDelegate {

    var scnView:ARSCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        let scene = SCNScene()

        let lightNode = SCNNNode()
        lightNode.light = SCNLight()
        lightNode.light!.type = .omni
        lightNode.position = SCNVector3(x: 0, y: 0, z: 0)
        scene.rootNode.addChildNode(lightNode)

        let ambientLightNode = SCNNNode()
        ambientLightNode.light = SCNLight()
        ambientLightNode.light!.type = .ambient
        ambientLightNode.light!.color = UIColor.darkGray
        scene.rootNode.addChildNode(ambientLightNode)

        let baseNode = SCNNNode()
        baseNode.position = SCNVector3(0, 0, -15)
        scene.rootNode.addChildNode(baseNode)

        let sunNode = SCNNNode()
        let sun = SCNSphere(radius: 2.5)
        sun.firstMaterial?.diffuse.contents = UIImage(named:"sun.jpg")
        sunNode.geometry = sun
        sunNode.position = SCNVector3(0, 0, 0);
```

```
let sunAnimation = CABasicAnimation(keyPath: "contentsTransform")
sunAnimation.duration = 10
sunAnimation.fromValue = NSValue.init(caTransform3D:
CATransform3DConcat(CATransform3DMakeTranslation(0, 0, 0),
CATransform3DMakeScale(3, 3, 3)))
sunAnimation.toValue = NSValue.init(caTransform3D:
CATransform3DConcat(CATransform3DMakeTranslation(1, 1, 0),
CATransform3DMakeScale(3, 3, 3)))
sunAnimation.repeatCount = .infinity

sun.firstMaterial?.diffuse.addAnimation(sunAnimation, forKey: nil)
sun.firstMaterial?.multiply.addAnimation(sunAnimation, forKey: nil)

baseNode.addChildNode(sunNode)

sun.firstMaterial?.multiply.contents = UIImage(named:"sun.jpg")
sun.firstMaterial?.multiply.intensity = 0.5;
sun.firstMaterial?.lightingModel = .constant

sun.firstMaterial?.diffuse.wrapS = .repeat
sun.firstMaterial?.diffuse.wrapT = .repeat
sun.firstMaterial?.multiply.wrapS = .repeat
sun.firstMaterial?.multiply.wrapT = .repeat

let earthRotationNode = SCNNode()
sunNode.addChildNode(earthRotationNode)

let earthGroupNode = SCNNode()
earthGroupNode.position = SCNVector3(15, 0, 0)
earthRotationNode.addChildNode(earthGroupNode)

let earthNode = SCNNode()
let earth = SCNSphere(radius: 1.5)
earth.firstMaterial?.diffuse.contents = UIImage(named:
"earth-diffuse-mini.jpg")
earthNode.geometry = earth
earthNode.position = SCNVector3(0, 0, 0)
earthGroupNode.addChildNode(earthNode)

let moonRotationNode = SCNNode()
earthGroupNode.addChildNode(moonRotationNode)

let moonNode = SCNNode()
let moon = SCNSphere(radius: 0.75)
moon.firstMaterial?.diffuse.contents = UIImage(named:"moon.jpg")
```

```
moonNode.geometry = moon
moonNode.position = SCNVector3Make(5, 0, 0);
moonRotationNode.addChildNode(moonNode)

let sunHalo = SCNPlane(width: 30, height: 30)
sunHalo.firstMaterial?.diffuse.contents = UIImage(named: "sun-halo.png")
sunHalo.firstMaterial?.emission.contents = UIImage(named: "sun-halo.png")
sunHalo.firstMaterial?.lightingModel = .constant
sunHalo.firstMaterial?.writesToDepthBuffer = false

let sunHaloNode = SCNNNode()
sunHaloNode.opacity = 0.4;
sunHaloNode.constraints = [SCNBillboardConstraint()]
sunHaloNode.geometry = sunHalo

sunNode.addChildNode(sunHaloNode)

earthRotationNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFloat(Float.pi * 2), z: 0,
        duration: 10)
    )
)

earthNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFloat(Float.pi * 2), z: 0,
        duration: 1)
    )
)

moonRotationNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFloat(Float.pi * 2), z: 0,
        duration: 1.5)
    )
)

moonNode.runAction(
    SCNACTION.repeatForever(
        SCNACTION.rotateBy(x: 0, y: CGFloat(Float.pi * 2), z: 0,
        duration: 1.5)
    )
)
```

```
scnView = ARSCNView()
self.view.addSubview(scnView)

scnView.translatesAutoresizingMaskIntoConstraints = false

self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "V:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "H:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))

scnView.scene = scene
scnView.allowsCameraControl = true
scnView.showsStatistics = true
scnView.backgroundColor = UIColor.black
}

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)

    let configuration = ARWorldTrackingConfiguration()

    scnView.session.run(configuration)
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    scnView.session.pause()
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
}

func session(_ session: ARSession, didFailWithError error: Error) {}
func sessionWasInterrupted(_ session: ARSession) {}
func sessionInterruptionEnded(_ session: ARSession) {}

PlaygroundPage.current.liveView = GameViewController()
PlaygroundPage.current.needsIndefiniteExecution = true
```

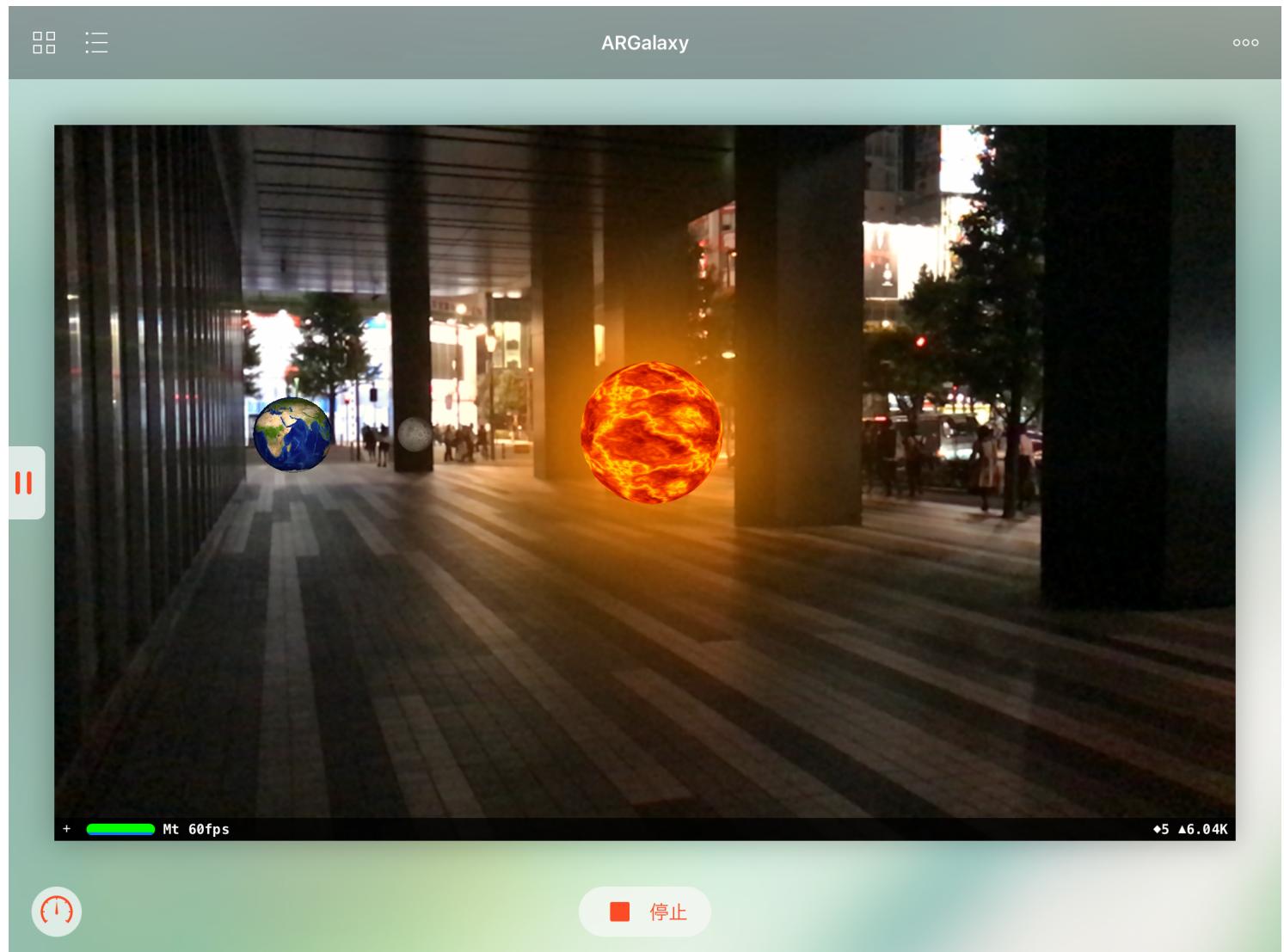


図 10.2 ARKit のサンプル 2

中央を左にスワイプするとライブビューの画面が広がります。

(縦の場合は上にスワイプ)

# 第 11 章

## シーン

---

シーンは実写の映像作成でいうなら撮影現場のようなもので、ここにカメラやライトや役者や小道具を配置することで映像の作成準備ができたこととなります。

iOS アプリの UIViewController に SceneKit の表示を設定する SCNView ですが、その SCNView に SCNScene に設定することによって、初めて 3DCG の仮想空間の初期設定が完成します。

また、本書でシーンと呼ぶ際は SCNScene のことを含みます。

### 11.1 SCNScene の役割

画面全体の設定をすることが SCNScene 役目となり、以下のものが設定できる項目です。

- ・シーンの初期化
- ・シーンの一時停止
- ・シーンの背景色、背景画像の設定
- ・背景画像を照明にするイメージベースドライティングの設定
- ・画面の遠方に霧の効果を与えるフォグ
- ・シーン全体の物理アニメーション設定
- ・シーンのアトリビュート設定
- ・シーンファイルの書き出し

これまで書いてきたように使用する際 SCNView の scene に SCNScene を設定します。本書では行いませんが、SCNView の scene に、他のシーンを設定することでシーンの切り替えが行え、ゲームなどの場面転換に使用できます。

## 11.2 シーンの初期化

そのまま初期化し、空のシーンを作成していましたが、SCNScene の初期化は以下のものがあります。

- 空のシーンを作成する
- シーンファイル (.scn) やオブジェクトファイルを読み込む
- Model I/O で設定したアセットから呼び出す

シーンファイルやオブジェクトファイルは URL を指定しネットワークから読み込みと、options を設定することでファイルの読み込む内容を設定することができ、シーンファイルからアニメーションだけ取り出すなどの指定ができます。

## 11.3 シーンファイルを読み込む

.scn シーンファイルを読み込みます。

リスト 11.1: シーンファイルを読み込みこむ

```
let scene = SCNScene(named: "scene.scn")!
```

## 11.4 シーンの一時停止設定

アクションなどのアニメーション、パーティクルアニメーション、物理アニメーションなどアニメーション全てを一時停止、再開を行います。

また、現在の一時停止、再開を行います。

リスト 11.2: 全体のアニメーションを止める

```
// SCNScene
let scene = SCNScene()

// isPaused = true で全体のアニメーションを止める
scene.isPaused = true
```

### 11.5 シーンの背景色、背景画像の設定

`background` でシーンの背景を設定します。

背景の色や画像、キューブマップテクスチャ、動画などを使用できます。

リスト 11.3: 背景色を設定する

```
// SCNScene
let scene = SCNScene()
scene.background.contents = UIColor.red
```

リスト 11.4: 背景画像を設定

```
// SCNScene
let scene = SCNScene()
scene.background.contents = UIImage(named: "texture.png")
```

#### 11.5.1 キューブマップの背景

キューブマップとは、立方体 6 面の内側に指定された画像を使用し 360 度の背景画像をつくり適応し、フリックすると周囲に設定したテクスチャが確認できます。コードは背景画像の適応方法と変わりません。キューブマップは Metal のみ使用可能で OpenGL では画像を平面に貼り付けた表示になります。

キューブマップの詳しい説明は別の章で紹介します。

#### 11.5.2 動画の背景

SpriteKit の `SKVideoNode` から、AVFoundation を使用し動画を再生することができます。

## 11.6 背景画像を照明にするイメージベースドライティングの設定

背景画像を照明にするイメージベースドライティングの画像を設定することによって、シーン上のジオメトリのマテリアルに光を与えますが、実際の動作はジオメトリのマテリアルに背景画像を元に色を与えています。

背景画像と同じ設定方法となっており、png などの標準的な画像以外にも.exr など HDR 情報を含んだファイルも使用できます。

### 11.6.1 注意点

- ・背景画像の照明として使用する画像はキューブマップに対応したものである
- ・適応するジオメトリのマテリアルのライティングモデルは Physically Based に設定されている。

リスト 11.5: イメージベースドライティング

```
// SCNScene
let scene = SCNScene()

scene.lightingEnvironment.contents = UIImage(named: "cubemap.png")
scene.lightingEnvironment.intensity = 1.0
```

## 11.7 シーンでの他の項目

シーン全体の物理アニメーション設定は物理アニメーションの章、フォグに関しては別の章でご紹介します。

アトリビュート設定は設定値がうまく動作せず、シーンファイルの書き出しがコードを書く量がかなり多くなるため、本書では割愛させていただきます。

# 第 12 章

## 名前

SceneKit のシングラフに配置できるものに関しては「name」のプロパティで名前を設定することができます。

### 12.1 名前をつける意味

名前をつけることで、シングラフから名前を元にオブジェクトの検索を行うことができます。主に、シーンファイルやプログラム的に複数ジオメトリを作成した際など、簡単にオブジェクトが指定できない場合で名前を元に調べることができます。

### 12.2 使用例

rootNode やチルドノードから childNode() を使い検索します。

以下のコードは、Xcode の Game テンプレートで使用されているもので、SCNView に設定された SCNScene のシングラフの起点 rootNode から「ship」の名前のノードを探し Y 軸 方向に回転させ他コードです。

リスト 12.1: ship のノードを探す

```
let ship = scene.rootNode.childNode(withName: "ship", recursively: true)!  
  
ship.runAction(SCNAction.repeatForever(SCNAction.rotateBy(x: 0, y: 2, z: 0,  
duration: 1)))
```

childNode の recursively: true は現在のノードの下の階層を調べるという設定です。

この場合で、false にした場合は rootNode より下の階層に「ship」の名前があっても検索対象になりません。

# 第13章

## カテゴリービットマスク

SceneKit のノードにはカテゴリービットマスク (CategoryBitmask) というものが存在おり、ノードや他のオブジェクトと判別するためにカテゴリービットマスクという整数の値があります。

### 13.1 カテゴリービットマスクの振る舞い

カテゴリービットマスクは対象に対して AND 演算をして 0 になった場合適応されます。デフォルトは 0 となっており適応され全て対象になります。また、マイナス値も 0 と同様になります。

ノードに対して、カメラに表示する、ライトを当てるような効果や、物理アニメーションで障害物の判定などを行います。

### 13.2 カメラでの使用例

例えば、球のジオメトリをカメラに表示したくない場合は、カメラのカテゴリービットマスクより上の値、または同じ値を球のジオメトリのカテゴリービットマスク設定します。

リスト 13.1: カテゴリービットマスクを使用して球を非表示にする

```
// カメラ
let cameraNode = SCNNNode()
cameraNode.camera = SCNCamera()
cameraNode.position = SCNVector3(x: 0, y: 0, z: 15)
cameraNode.camera?.categoryBitMask = 1
scene.rootNode.addChildNode(cameraNode)

// 球のジオメトリ
let sphere = SCNSphere(radius: 1.0)
let sphereNode = SCNNNode(geometry:sphere)
```

```
sphereNode.categoryBitMask = 2  
scene.rootNode.addChildNode(sphereNode)
```

SCNCamera のカテゴリービットマスクを設定しているのは、デフォルトでは 0 であるため、球のジオメトリのカテゴリービットマスクを変更しても適応されないためです。

# 第14章

## ノードについて (SCNNode)

ノードは SceneKit の仮想空間で位置、回転、拡大縮小やその他の情報を表すシーディングラフの構造要素で目に見えるコンテンツを持ちません。

ノードの下に、ライト、カメラ、ジオメトリ、アニメーションなど様々な機能を紐付けて目に見えるコンテンツを作成します。

### 14.1 移動、回転、拡大縮小

何回か使用していますが、ルートノードを除くすべてのノードで移動、回転、拡大縮小が行え、クオータニオンでの回転と行列での移動、回転、拡大縮小が可能です。

#### 14.1.1 移動

SCNNode の position に SCNVector3 を渡して位置を変更し移動させます。SCNVector3 は X、Y、Z の順です。

リスト 14.1: 指定位置に移動

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

// X 軸 +2 方向に位置を設定し移動
boxNode.position = SCNVector3(2, 0, 0)

scene.rootNode.addChildNode(boxNode)
```

### 14.1.2 回転

回転方法は以下の3つです。

- 3つのベクトルを使用する SCNVector3 で SCNNode の eulerAngles に渡す
- 4つのベクトルを使用する SCNVector4 で SCNNode の rotation に渡す
- 4つのベクトルを使用する SCNQuaternion で SCNNode の orientation に渡す

また、回転を渡す値は、ラジアンです。1  $\pi$  rad が 180 度ですので、以下のサンプルではその 1/4 の 0.25  $\pi$  rad で回転しています。

#### eulerAngles

設定する数が少ないので、こちらを主に使用することになると思います。position と同じで SCNVector3 は X、Y、Z です。

リスト 14.2: eulerAngles で回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

// X 軸に 45 度の回転
boxNode.rotation = SCNVector3(0.25 * Float.pi, 0, 0)

scene.rootNode.addChildNode(boxNode)
```

#### rotation

SCNVector4 は X、Y、Z、W で W の値を X、Y、Z へ影響させます。以下の例の SCNVector4 の X を 0.5 にすると半分の 22.5 度回転します。

リスト 14.3: rotation で回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

// X 軸に 45 度の回転
```

```
boxNode.rotation = SCNVector4(1, 0, 0, 0.25 * Float.pi)  
scene.rootNode.addChildNode(boxNode)
```

以下の例は、X 軸方向に 1、Z 軸方向に 2 が設定されているため、X に 45 度の回転に加え、Z 軸に 90 度の回転が加わります。

リスト 14.4: 他の軸も rotation で回転

```
boxNode.rotation = SCNVector4(1, 0, 2, 0.25 * Float.pi)
```

### orientation

orientation に SCNQuaternion (SCNVector4) を渡して回転を行い、rotation 同様 SCNVector4 ですが、こちらは  $xx + yy + zz + ww == 1$  を満たすクオータニオンでノードを回転させます。

例は記載しますが、ジンバルロックなど説明が多いため、こちらの説明は省きます。

リスト 14.5: クオータニオンで回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)  
let boxNode = SCNNNode(geometry: box)  
  
// X 軸に 45 度の回転  
boxNode.orientation = SCNVector4(x: 0.382683426141739, y: 0.0, z: 0.0,  
w: 0.923879563808441)  
  
scene.rootNode.addChildNode(boxNode)
```

使用頻度は少ないと思いますが、クオータニオンに変換する関数を書きました。angle でラジアン、SCNVector3 で適応角度を使用し回転させます。

リスト 14.7: ラジアンをクオータニオンで返す関数

```
func toQuaternion(angle: Float, axis: SCNVector3) -> SCNQuaternion {  
    let half = angle * 0.5  
    let sin_theta = CGFloat(sin(half))  
  
    var x = axis.x  
    var y = axis.y  
    var z = axis.z  
    let w = CGFloat(cos(half))  
  
    let length = 1.0 / sqrt(x*x + y*y + z*z)  
  
    x = x * length;  
    y = y * length;  
    z = z * length;  
  
    x = x * sin_theta  
    y = y * sin_theta  
    z = z * sin_theta  
  
    return SCNQuaternion.init(x, y, z, w)  
}
```

リスト 14.7: クオータニオンで X 軸 45 度回転

```
boxNode.orientation = toQuaternion(angle: 0.25 * Float.pi, axis:  
SCNVector3(1, 0, 0))
```

### 14.1.3 拡大縮小

`position` と同様に `SCNVector3` を使用し、初期値は X, Y, Z 共に 1 です。下の例では X, Y, Z で 2 を指定し、2 倍の大きさにしており、今回はプラス値を設定していますがマイナス値で縮小します。

リスト 14.10: 2 倍に拡大

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

// 2倍に拡大
boxNode.scale = SCNVector3(2, 2, 2)

scene.rootNode.addChildNode(boxNode)
```

## 14.2 ピボットの変更

ノードにはオブジェクトの基準位置を決めるピボットというものがあります。

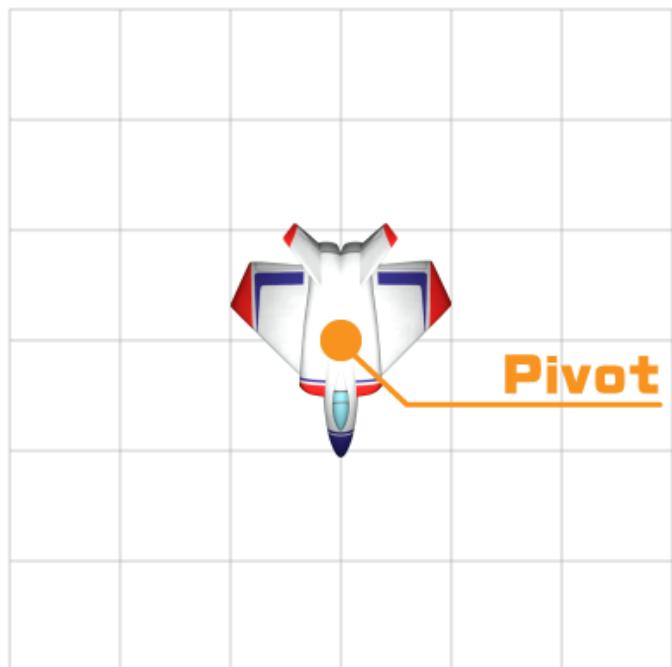
球のジオメトリである SCNSphere は Pivot が中心にあり、回転を行なった場合、通常ならジオメトリ中心から回転動作が行われますが、ピボットを X 軸方向 2 に移動し変更すると円を描いて回転します。

リスト 14.9: Pivot の変更と動作

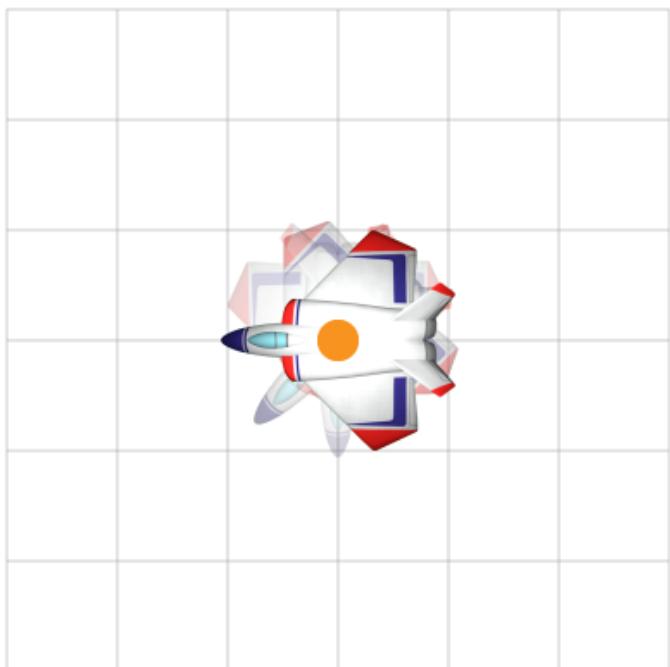
```
// シーンからノードを検索
let ship = scene.rootNode.childNode(withName: "ship", recursively: true)!

// pivot 変更
ship.pivot = SCNMatrix4MakeTranslation(2.0, 0.0, 0.0)
```

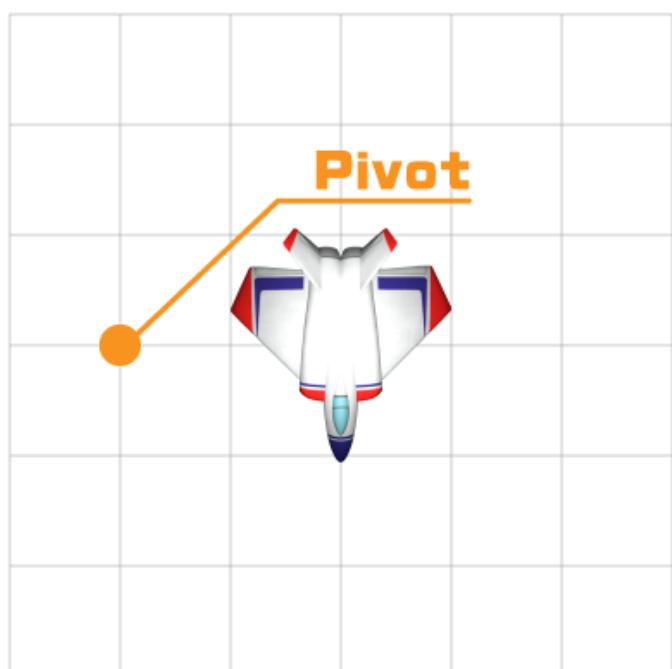
Pivot X 軸 0



Y 軸 -90 度回転



Pivot X 軸 +2



Y 軸 -90 度回転

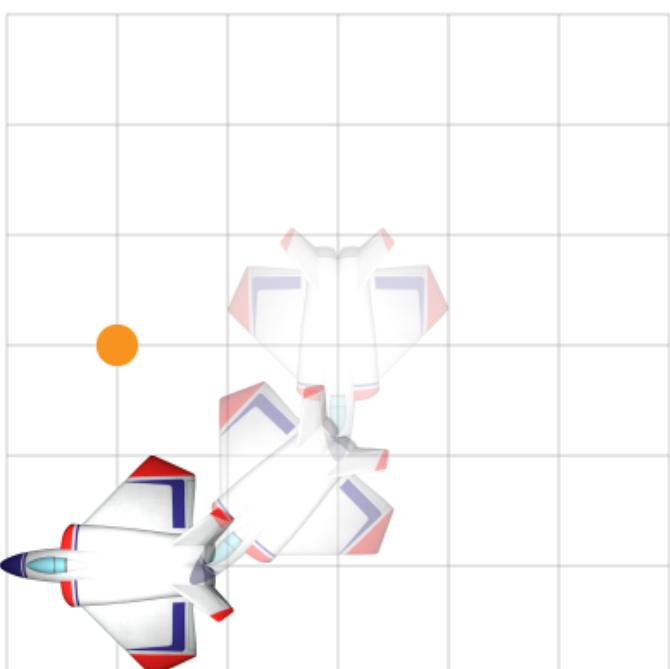


図 14.1 ピボットの変更と動作

## 14.3 透明度

ノードとそのノードに設定されているチルドノードの全ての透明度を設定することができ、初期値は 1 となっており、0 で完全に透明になります。

チルドノードなどを透明にしたくない場合は、マテリアルの透明度の方で設定します。

リスト 14.10: 2 倍に拡大

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

// 50 %透明にする
boxNode.opacity = 0.5

scene.rootNode.addChildNode(boxNode)
```

## 14.4 表示、非表示

透明にする場合はジオメトリが透明になるだけで、ジオメトリ自体は存在しています。非表示にした場合、カメラやライトの適応範囲から外され、その存在がシーンから表示されなくなります。

リスト 14.12: 非表示にする

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

// 非表示にする
boxNode.isHidden = true

scene.rootNode.addChildNode(boxNode)
```

表示がされなくなるため、レンダリング処理をしたり、ドローコールは増えませんが、シ-

シングラフには存在しています。そのため、非表示にしている間にマテリアルの変更をしたり、シングラフからノードを名前を検索など行うことができます。

### 14.4.1 レンダリングオーダー

シーンを描画する順番を決めます。

リスト 14.12: 描画する順番を決める

```
boxNode.renderingOrder = 5
```

### 14.4.2 Movability Hint

Light Probe でジオメトリのマテリアルの Emission を照明にする際使用します。

リスト 14.13: Movability Hint

```
boxNode.movabilityHint = true
```

### 14.4.3 キャストシャドウ

他のジオメトリから落ちる影を表示するかしないかを決めます。初期値は `true` になっていますが、もし必要がなければ `false` にした方が処理は速くなります。

リスト 14.14: Casts Shadow

```
boxNode.castsShadow = true
```

## 14.5 ノード単位で再生を停止する

ノードとそのチルドノードのアニメーションとアクションを停止します。物理アニメーションやパーティクルはアニメーションが止まらないので注意が必要です。

リスト 14.15: Paused

```
boxNode.isPaused = true
```

# 第 15 章

## 行列を使用したノードの移動、回転、拡大縮小

---

行列で使用する SCNMatrix4 は  $4 \times 4$  の Float の行列で、設定値が 16 個あり複雑です。興味がなければこの章は読み飛ばしてもらって構いません。

機能によっては、SCNMatrix4 を使用しなければいけないものがありますが、もし使用する場合は、この章の後半で簡単に使用する命令があるため、そちらを使用したほうがよさそうです。

### 15.1 SCNMatrix4 の初期設定

まずは SCNMatrix4 の初期状態を設定。行列の対角要素を 1.0 になっており、他を 0.0 で埋めた行列になっています。

リスト 15.3: 初期設定

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

let m = SCNMatrix4(
    m11: 1.0, m12: 0.0, m13: 0.0, m14: 0.0,
    m21: 0.0, m22: 1.0, m23: 0.0, m24: 0.0,
    m31: 0.0, m32: 0.0, m33: 1.0, m34: 0.0,
    m41: 0.0, m42: 0.0, m43: 0.0, m44: 1.0
)
boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

## 15.2 行列を使用し拡大縮小させる

上の例では SCNMatrix4 で 16 個設定して初期設定をしていますが、面倒なので SCNMatrix4Identity で同じものをつくります。

SCNMatrix4 の m11 は X、m22 は Y、m33 は Z のスケールに対応しており、m11、m22、m33 へ 2 を個別に設定し 2 倍の大きさに変更します。

リスト 15.2: 拡大縮小させる

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

var m = SCNMatrix4Identity
m.m11 = 2.0
m.m22 = 2.0
m.m33 = 2.0

boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

## 15.3 行列を使用し移動させる

X は m41、Y は m42、Z は m43 に対応しており、下の例では X 軸に移動します。

リスト 15.3: 行列を使用し移動させる

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

var m = SCNMatrix4Identity
m.m41 = 2.0
m.m42 = 0.0
m.m43 = 0.0

boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

## 15.4 行列を使用し回転させる

行列の回転は拡大縮小や移動と比べ複雑で、決められた場所に  $\sin$  と  $\cos$  を使用します。下の例では、X、Y、Z の各軸に 45 度の回転を入れています

### 15.4.1 X 軸回転

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

リスト 15.6: X 軸回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

var m = SCNMatrix4Identity

m.m22 = cos(0.25 * Float.pi)
m.m23 = sin(0.25 * Float.pi)
m.m32 = -sin(0.25 * Float.pi)
m.m33 = cos(0.25 * Float.pi)

boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

## 15.4.2 Y 軸回転

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

リスト 15.6: X 軸回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

var m = SCNMatrix4Identity

m.m11 = cos(0.25 * Float.pi)
m.m13 = sin(0.25 * Float.pi)
m.m31 = -sin(0.25 * Float.pi)
m.m33 = cos(0.25 * Float.pi)

boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

## 15.4.3 Z 軸回転

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

リスト 15.6: Z 軸回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

var m = SCNMatrix4Identity
```

```
m.m11 = cos(0.25 * Float.pi)
m.m12 = sin(0.25 * Float.pi)
m.m21 = -sin(0.25 * Float.pi)
m.m22 = cos(0.25 * Float.pi)

boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

### 15.5 もっと簡単に設定したい

`SCNMatrix4MakeScale`、`SCNMatrix4MakeTranslation`、`SCNMatrix4MakeRotation` というものが用意されており、`SCNVector3`、`4` で設定しているように変更できるものが用意されています。

#### 15.5.1 SCNMatrix4MakeScale

リスト 15.7: 2 倍拡大

```
boxNode.transform = SCNMatrix4MakeScale(2, 2, 2)
```

#### 15.5.2 SCNMatrix4MakeTranslation

リスト 15.9: X 軸 2 移動

```
boxNode.transform = SCNMatrix4MakeTranslation(2, 0, 0)
```

#### 15.5.3 SCNMatrix4MakeRotation

最初の引数が角度になっているので注意してください。

### リスト 15.9: X 軸 45 度回転

```
boxNode.transform = SCNMatrix4MakeRotation(0.25 * Float.pi, 1, 0, 0)
```

## 15.6 簡単に移動、回転、拡大縮小と一緒に使いたい

SCNMatrix4Translate、SCNMatrix4Rotate、SCNMatrix4Scale では、SCNMatrix4 に対して移動、回転、拡大縮小を合成します。

以下のコードでは X 軸に 45 度回転、2 倍拡大し、X 軸を 2 に移動しています。

### リスト 15.10: 回転、拡大、移動

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

var m = SCNMatrix4Identity

m = SCNMatrix4Rotate(m, 0.25 * Float.pi, 1, 0, 0)
m = SCNMatrix4Scale(m, 2, 2, 2)
m = SCNMatrix4Translate(m, 2, 0, 0)

boxNode.transform = m

scene.rootNode.addChildNode(boxNode)
```

# 第 16 章

## ジオメトリ

---

SceneKit ではシーンに設置される形のある物体をジオメトリと呼び、この章では ジオメトリの概要と SceneKit で用意されたビルトインジオメトリ、次章ではカスタムジオメトリについて紹介します。

### 16.1 ジオメトリの構成

ジオメトリの表面はポリゴン（多角形）で構成されています。ポリゴンとなると 3 角形以上の面が複数で構成されており、面にはそれを構成する頂点が必要となります。

面を構成するために裏か表かを判別するため法線が必要になり、テクスチャを貼り付ける場合は UV の情報を付加することでジオメトリが構成されます。<sup>\*1</sup>

### 16.2 ビルトインジオメトリの利点

3DCG のオーサリングツール（DCC ツール）や自分でプログラムからジオメトリを制作しなくても、手軽に作成することができます。また、UV マップが作成済みですのでそのままテクスチャを貼ることができます。

---

<sup>\*1</sup> その他に頂点カラーとスキニングアニメーションのウェイトが設定できます

## 16.3 SceneKit で用意されているビルトインジオメトリ

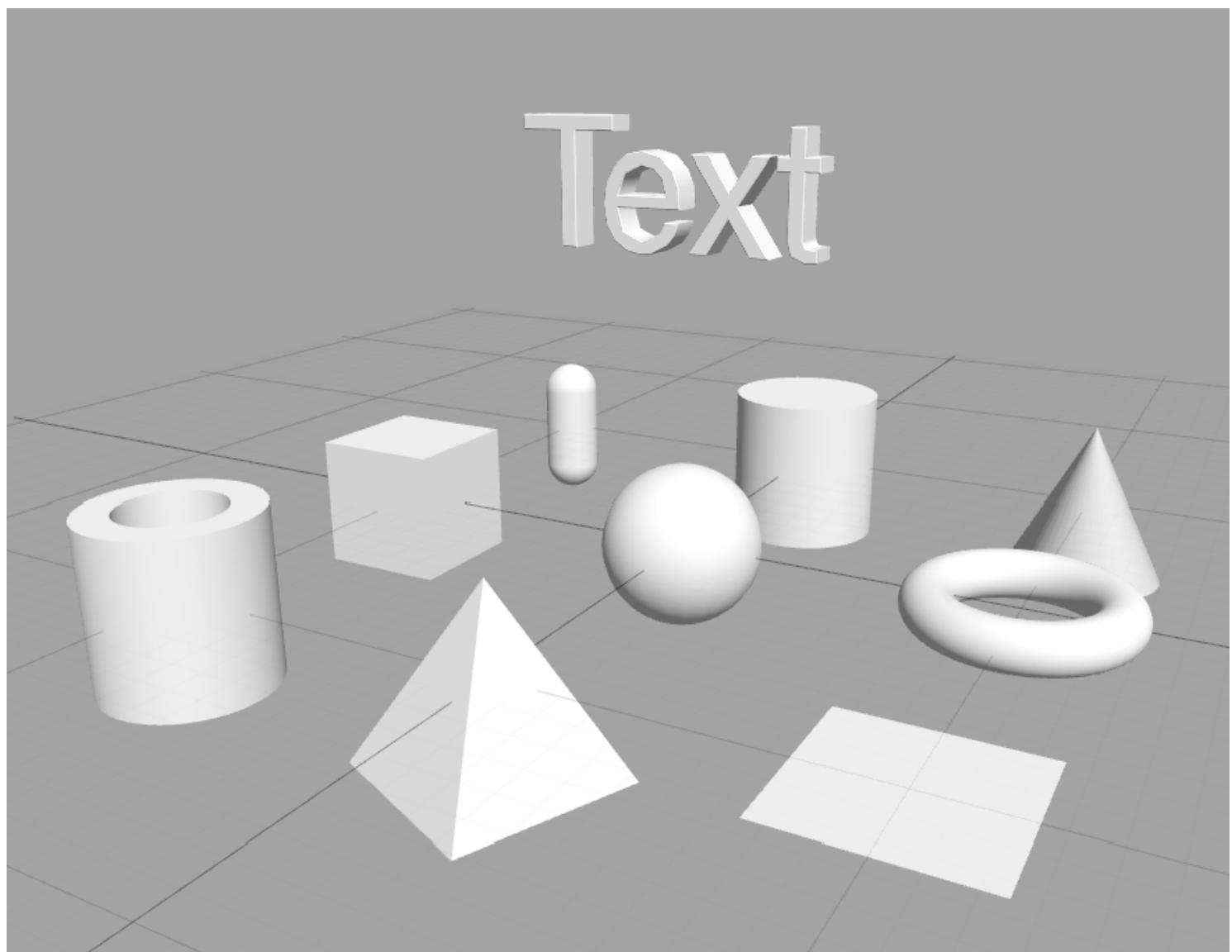


図 16.1 ビルトインジオメトリ

すでに SCNSphere と SCNPlane は使用していますが、SceneKit で用意されているビルトインジオメトリは以下のものとなります。

- 無限平面 (SCNFloor)
- 立方体 (SCNBox)
- カプセル型 (SCNCapsule)
- 円錐 (SCNCone)

- 円柱 (SCNCylinder)
- 平面 (SCNPlane)
- 三角錐 (SCNPyramid)
- 球 (SCNSphere)
- ドーナツ型 (SCNTorus)
- チューブ型 (SCNTube)
- テキスト (SCNText)
- パスからの図形 (SCNShape)

## 16.4 無限平面 (SCNFloor)

初期値のまま原点に配置すると、Y 軸 0、XZ 軸が無限の大きさを持つ平面を作成します。

### Reflectivity

0～1 の範囲の値で反射率を表します。1 が完全に反射し 0 が反射なしとなり、初期値は 0.25 です。

### Falloff start / Falloff end

反射の減退の開始と終わりを設定します。初期値は共に 0 で減退しません。start、end の間が少なすぎると急に減退し違和感が出るので注意。

### Resolution factor

鏡面反射はシーンがオフスクリーン（画面が描画される前）で 1 度レンダリングされ、さらに床のテクスチャーをレンダリングして、最終的にそれらを合わせたものを画面にレンダリングします。この値は鏡面反射のレンダリング画質で 0～1 の間で設定します。

デフォルトは iOS は 0.5、macOS は 1 となっており、iOS は負荷が少ないように設定されています。

### サンプルコード

リスト 16.1: SCNFloor

```
// SCNFloor
let floor = SCNPlane()
floor.reflectivity = 0.25
floor.reflectionFalloffStart = 0
floor.reflectionFalloffEnd = 0
floor.reflectionResolutionScaleFactor = 0.5

let floorNode = SCNNNode(geometry: floor)
scene.rootNode.addChildNode(floorNode)

// 鏡面反射用の球
let ball = SCNSphere(radius: 0.5)
let ballNode = SCNNNode(geometry: ball)
ballNode.position = SCNVector3(0,1,0)
scene.rootNode.addChildNode(ballNode)
```

## 16.5 立方体 (**SCNBox**)

立方体のジオメトリを作成します。角を丸くすることもできます。

### Size

`width`、`height`、`length` で立方体の大きさを設定します。初期値はすべて 1。

width	幅
height	高さ
length	奥行き

### Chamfer radius

角丸の半径の値で、初期値は 0。`width`、`height`、`length` が 1 で `Chamfer radius` 1 の場合は球体になります。

`Chamfer radius` を設定するとジオメトリの構造が変わるのでテクスチャを貼る際など注意が必要です。

### Segment Count

面のポリゴンの分割数を `width`、`height`、`length` で設定し、初期値は 1 で Int 型です。

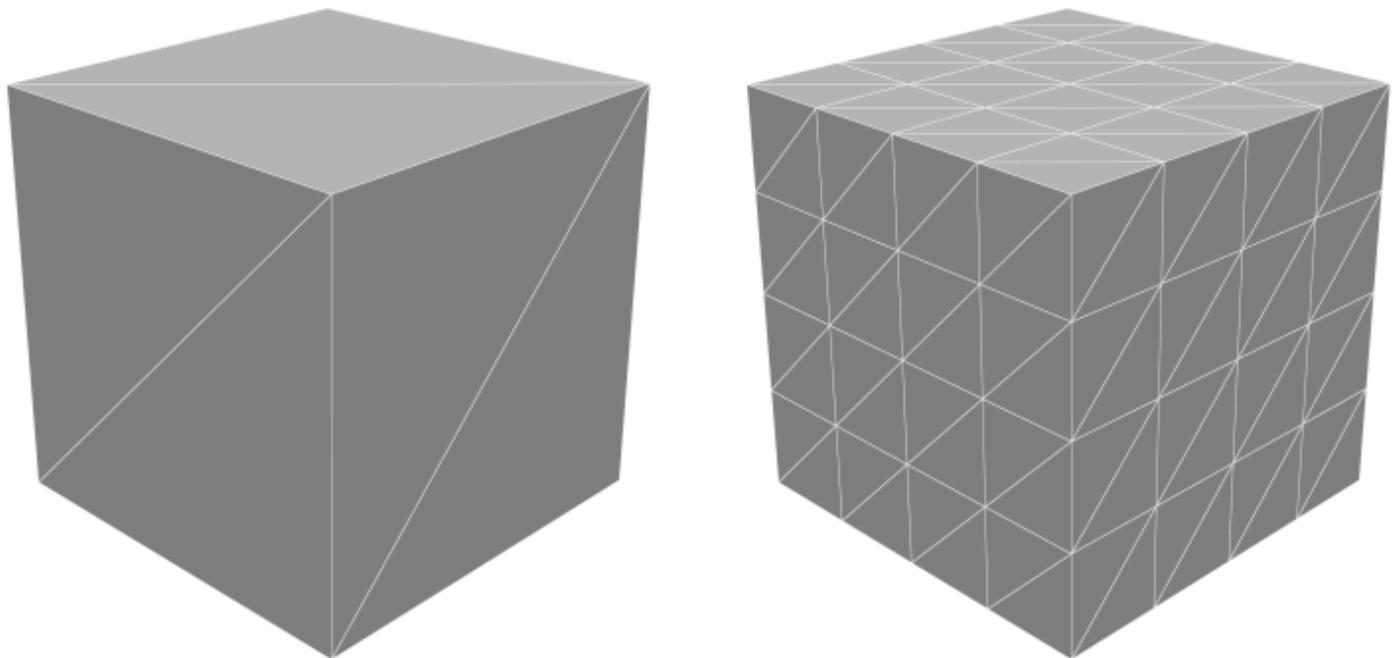


図 16.2 Segment Count を 1 と 4 で設定

### Chamfer Segment Count

角丸のポリゴンの分割数を指定します。初期値は 10 で Int 型です。

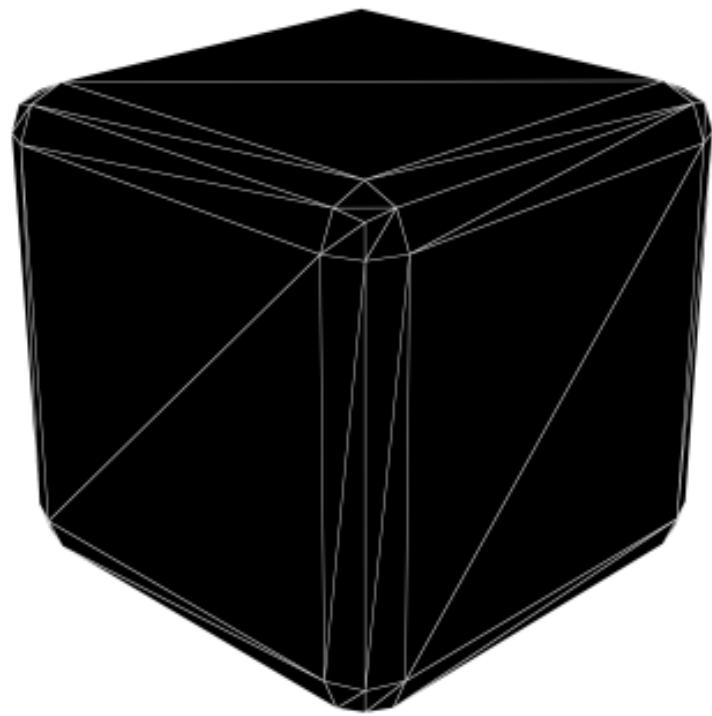


図 16.3 Chamfer Segment Count を 2 に設定

## サンプルコード

### リスト 16.2: SCNBox

```
// SCNBox
let box = SCNBox(width: 1, height: 1, length: 1, chamferRadius: 0.1)

box.widthSegmentCount = 4
box.heightSegmentCount = 4
box.lengthSegmentCount = 4
box.chamferSegmentCount = 2

let node = SCNNNode(geometry: box)
scene.rootNode.addChildNode(node)
```

## 16.6 カプセル型 (**SCNCapsule**)

円柱の上下に半球をつけたカプセル型のジオメトリを作成します。

### **Cap radius**

カプセルの半径、初期値は 0.25 です。

### **Cap radius**

カプセルの高さ（半球部分も含む）、初期値は 1 です。

高さは半球含むため、高さ 1、半径 0.5 にすると球体になります。

### **Radial Segment Count**

半球外周部分のポリゴンの分割数、初期値は 48 です。

### **Height Segment Count**

円柱の分割数、初期値は 1 です。

### **Cap Segment Count**

半球の高さ部分のポリゴンの分割数で、初期値は 24 です。

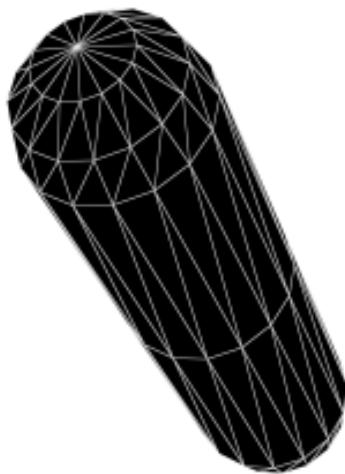


図 16.4 半球外周部分 16、半球の高さ 3、円柱の高さ 2 に設定

### サンプルコード

#### リスト 16.3: SCNCapsule

```
// SCNCapsule
let capsule = SCNCapsule(capRadius: 0.25, height: 2)

capsule.radialSegmentCount = 16
capsule.heightSegmentCount = 2
capsule.capSegmentCount = 3

let node = SCNNode(geometry: capsule)
scene.rootNode.addChildNode(node)
```

## 16.7 円錐 (SCNCone)

円錐のジオメトリを作成します。

### Top radius

上底の半径、初期値は 0 です。

Bottom radius と同じ値にすると円柱になります。

### Bottom radius

下底の半径、初期値は 0.5 です。

### Height

円錐の高さ、初期値は 1 です。

### Radial Segment Count

外周部分のポリゴンの分割数、初期値は 48 です。

### Height Segment Count

円錐の高さの分割数、初期値は 1 です。

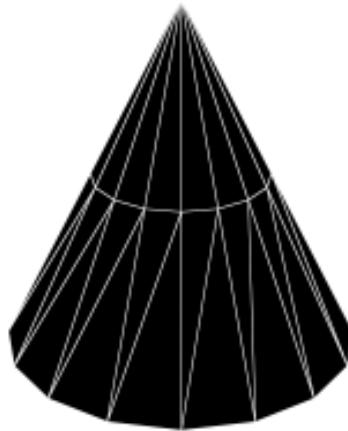


図 16.5 外周部分 16、高さ 2 に設定

#### リスト 16.4: SCNCone

```
// SCNCone
let cone = SCNCone(topRadius: 0, bottomRadius: 0.5, height: 1)

cone.radialSegmentCount = 16
cone.heightSegmentCount = 2

let node = SCNNNode(geometry: cone)
scene.rootNode.addChildNode(node)
```

## 16.8 円柱 (SCNCylinder)

円柱のジオメトリを作成します。

### Radius

底の半径、初期値は 0.5 です。

### Height

円柱の高さ、初期値は 1 です。

### Radial Segment Count

外周部分のポリゴンの分割数、初期値は 48 です。

### Height Segment Count

円柱の高さの分割数、初期値は 1 です。

リスト 16.5: SCNCylinder

```
// SCNCylinder
let cylinder = SCNCylinder(radius: 0.5, height: 1)

cylinder.radialSegmentCount = 48
cylinder.heightSegmentCount = 1

let node = SCNNNode(geometry: cylinder)
scene.rootNode.addChildNode(node)
```

## 16.9 平面 (SCNPlane)

奥行き 0 の縦状態がデフォルトの平面のジオメトリを作成します。また、四隅の角を丸く設定することができます。

### Size

幅と高さが設定できます。初期値は 1 です。

### Corner radius

角丸の大きさ、初期値は 0 です。

### Segment Count

幅と高さの分割数、初期値は 1 です。

### Corner

角丸の分割数、初期値は 10 です。

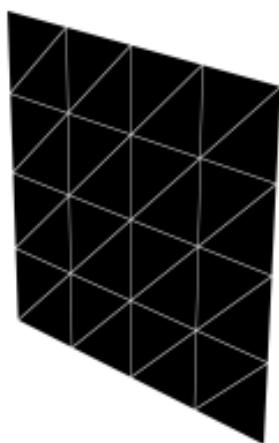


図 16.6 幅、高さの分割数を 4 に設定

### サンプルコード

リスト 16.6: SCNPlane

```
// SCNPlane
let plane = SCNPlane(width: 1, height: 1)

plane.cornerRadius = 0.2
plane.widthSegmentCount = 4
plane.heightSegmentCount = 4
plane.cornerSegmentCount = 4

let node = SCNNNode(geometry: plane)
scene.rootNode.addChildNode(node)
```

## 16.10 三角錐 (SCNPyramid)

三角錐のジオメトリを作成する。

### Size

幅、高さ、奥行きの設定で、初期値は 1 です。

### Segment Count

幅、高さ、奥行きの分割数で、初期値は 1 です。

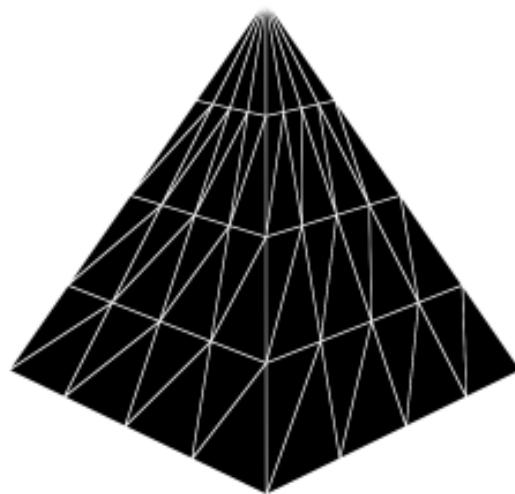


図 16.7 幅、高さ、奥行きの分割数を 4 に設定

## サンプルコード

## リスト 16.7: SCNPyramid

```
// SCNPyramid
let pyramid = SCNPyramid(width: 1, height: 1, length: 1)

pyramid.widthSegmentCount = 4
pyramid.heightSegmentCount = 4
pyramid.lengthSegmentCount = 4

let node = SCNNNode(geometry: pyramid)
scene.rootNode.addChildNode(node)
```

## 16.11 球 (SCNSphere)

球のジオメトリを作成します。

### Radius

球の半径で初期値は 1 です。半径であるため、ジオメトリは倍の大きさになります。

### Segment Count

X 軸と Y 軸方向での外周での分割数、初期値は 24 です。Geodesic のチェックが入っていると正 20 面体からの分割数となります。

### Geodesic

チェックすると三角ポリゴンを均等に面が構成されるされます。

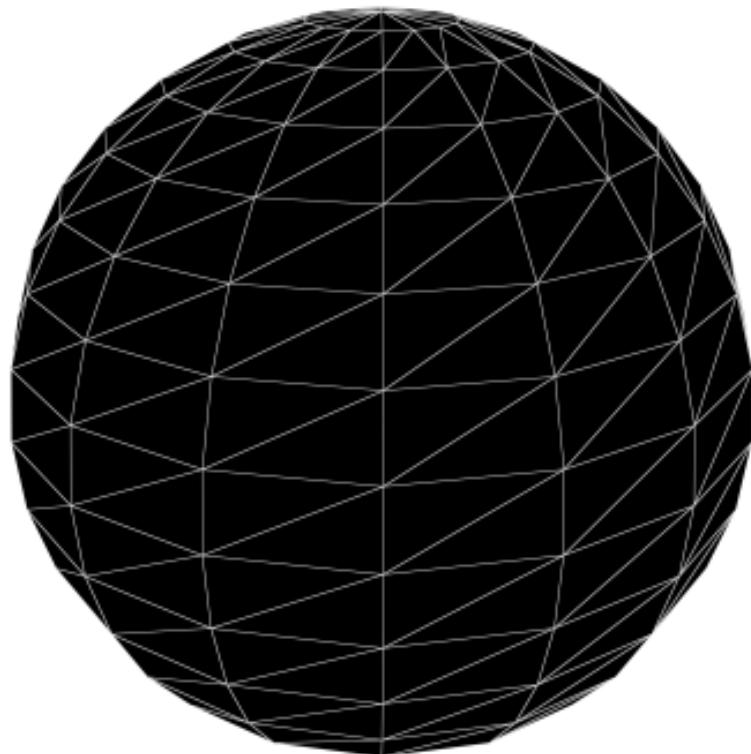


図 16.8 Geodesic を false、分割数を 16 に設定

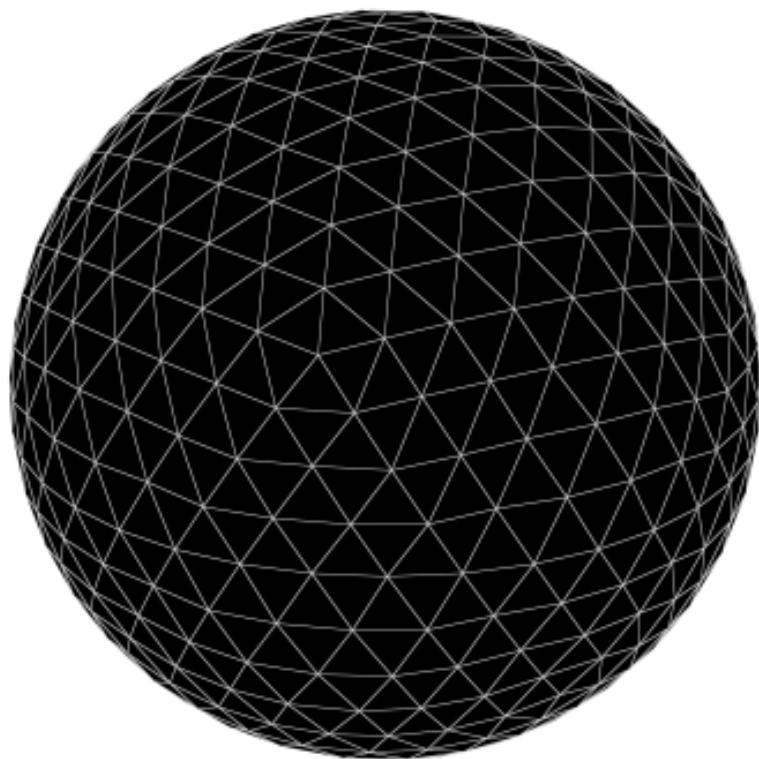


図 16.9 Geodesic を true、分割数を 16 に設定

### サンプルコード

#### リスト 16.8: SCNSphere

```
// SCNSphere
let sphere = SCNSphere(radius: 1)

sphere.segmentCount = 16
sphere.isGeodesic = false

let node = SCNNode(geometry: sphere)
scene.rootNode.addChildNode(node)
```

## 16.12 ドーナツ型 (**SCNTorus**)

ドーナツ型のジオメトリを作成します。

### Ring Radius

オブジェクト全体の半径、初期値は 1 です。半径であるため、ジオメトリは倍の大きさになります。

### Pipe Radius

ドーナツのパイプ状になっている部分の半径、初期値は 0.25 です。値を大きくするとリング部分が大きくなり、穴は小さくなります。

### Ring Segment Count

外周の分割数、初期値は 48 です。

### Pipe Segment Count

ドーナツのパイプ状になっている部分の分割数、初期値は 24 です。

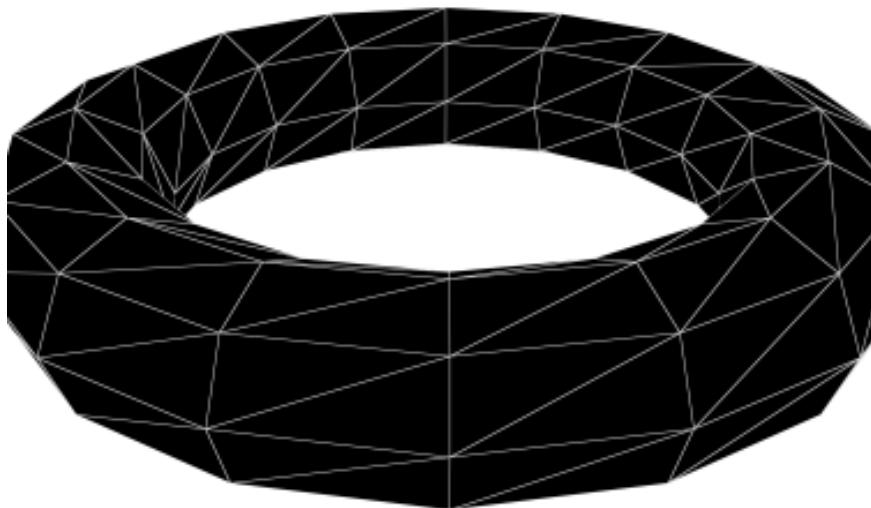


図 16.10 外周を 16、パイプ部分を 8 を設定

## サンプルコード

### リスト 16.9: SCNTorus

```
// SCNTorus
let torus = SCNTorus(ringRadius: 1, pipeRadius: 0.25)

torus.ringSegmentCount = 16
torus.pipeSegmentCount = 8

let node = SCNNode(geometry: torus)
scene.rootNode.addChildNode(node)
```

## 16.13 チューブ型 (SCNTube)

チューブ型のジオメトリを作成します。

### Inner radius

チューブ内側の円の半径、初期値は 0.5 です。Outer radius と同じするとチューブの厚さがなくなります。

### Outer radius

外周の半径、初期値は 1 です。Inner radius と同じするとチューブの厚さがなくなります。

### Height

ジオメトリの高さ、初期値は 1 です。

### Radial Segment Count

外周の分割数、初期値は 48 です。

### Height Segment Count

高さの分割率、初期値は 1 です。

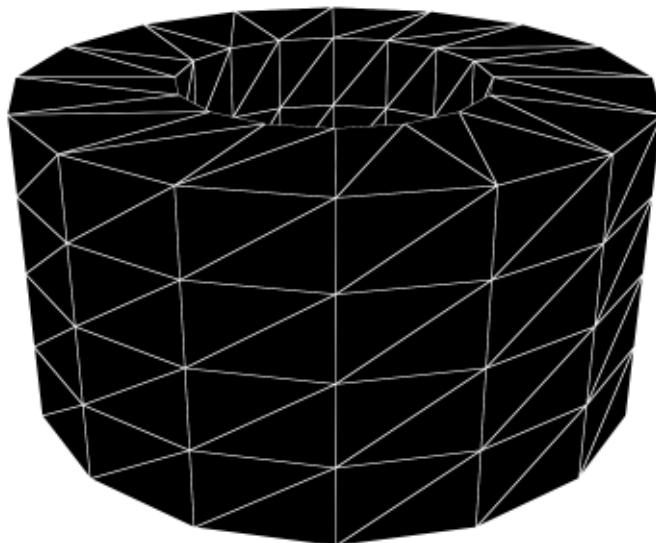


図 16.11 外周を 16、高さを 4 の分割数に設定

## サンプルコード

### リスト 16.10: SCNTube

```
// SCNTube
let tube = SCNTube(innerRadius: 0.5, outerRadius: 1, height: 1)

tube.radialSegmentCount = 16
tube.heightSegmentCount = 4

let node = SCNNode(geometry: tube)
scene.rootNode.addChildNode(node)
```

## 16.14 テキスト (SCNText)

NSString もしくは NSAttributedString で設定したテキストから 3D テキストのジオメトリを作成します。

ベースラインではなく、ディセンダ（下の空き）を含めるため、原点においても文字は左下

は原点にこないので注意が必要です。また、文字の奥行きが 0 の場合はポリゴンが表裏描画されます。

### Text

表示する文字列を設定します。

### Font

フォントとフォントの大きさの設定します。初期値は Helvetica のレギュラーで、サイズは 36 ポイントです。

### Extrusion depth

3D テキストの奥行き、初期値は 1 です。

### Flatness

3D テキストのカーブなど湾曲している部分のポリゴンの細かさ、初期値は 0.6 です。  
0 で最も細かくなるなります。

### Chamfer radius

3D テキスト角部分の面取りの大きさになります。初期値は 0 で面取りが行われません。

### Chamfer profile

パスから面取りの形を設定します。

### String

表示する文字列を設定します。NSAttributedString での設定も可能です。

### Font

UIFont を設定します

### containerFrame

CGRect で SCNText の大きさを設定できます。

### **isWrapped**

文字の折り返しを行うかを設定します。

### **truncationMode**

トランケート設定を設定します。

### **alignmentMode**

文字の向きを設定します。(右寄せ、中央、左寄せ、など)

### **textSize**

文字の大きさ設定します。ドキュメントにはこのプロパティが書いてありますが、SCNText の定義でこちらのプロパティを見つけることができませんでした。使用できるか不明です。

領域を設定し改行設定していると、設定した範囲に入るようになります。



図 16.12 containerFrame による表示領域指定

領域内に文字が入らないため、「…」でトランケート表示されます



図 16.13 truncationMode によるトランケート指定

角丸部分をパスで設定と色分けができます。



図 16.14 chamferProfile と マテリアル指定

### 注意点

- containerFrame で表示サイズを設定しない場合は isWrapped、truncationMode、alignmentMode は使用できません。
- NSAttributedString が設定されると font、isWrapped、truncationMode、alignmentMode 設定していても無視されます。自分でアトリビュートを設定する必要があります。

ます。

- chamferProfile は始点が (x: 0.0, y: 1.0)、終点 (x: 1.0, y: 0.0) の間で設定されている UIBezierPath を与えると描画されます。
- chamferRadius、chamferProfile などが正しく設定されていない状態で flatness を 0 にするとクラッシャーすることがあります。
- SCNText のマテリアルは [前面文字、前面 chamfer、伸びている部分、後面 chamfer、後面文字] の 5 つの配列で構成されます。指定されない場合は、通常どうり firstMaterial が適応されます。

## サンプルコード

リスト 16.12: SCNText

```
// SCNText
let text = SCNText(string: "SCNText!!!", extrusionDepth: 1)

// NSAttributedString 設定
let style = NSMutableParagraphStyle()
style.lineSpacing = -0.5

let textFont:UIFont = UIFont(name: "Futura-Bold", size: 0.5)!

text.string = NSAttributedString(string: "SCNText!!!", attributes: [
    NSParagraphStyleAttributeName: style,
    NSFontAttributeName: textFont
])

// その他の設定
text.flatness = 0.0
text.extrusionDepth = 1
text.chamferRadius = 0.02

// chamfer の形状設定
let path = UIBezierPath()
path.move(to: CGPoint(x: 0.0, y: 1.0))
path.addLine(to: CGPoint(x: 0.0, y: 0.75))
path.addLine(to: CGPoint(x: 0.25, y: 0.75))
path.addLine(to: CGPoint(x: 0.25, y: 0.5))
path.addLine(to: CGPoint(x: 0.5, y: 0.5))
path.addLine(to: CGPoint(x: 0.5, y: 0.25))
```

```
path.addLine(to: CGPoint(x: 0.75, y: 0.25))
path.addLine(to: CGPoint(x: 0.75, y: 0.0))
path.addLine(to: CGPoint(x: 1.0, y: 0.0))

text.chamferProfile = path

// マテリアル設定

let m1 = SCNMaterial()
let m2 = SCNMaterial()
let m3 = SCNMaterial()

m1.diffuse.contents = UIColor(red: 0.7, green: 0.7, blue: 0.7, alpha: 1.0)
m2.diffuse.contents = UIColor.red
m3.diffuse.contents = UIColor.yellow

text.materials = [m3,m2,m1,m2,m3]

let node = SCNNNode(geometry: text)
node.position = SCNVector3Zero
scene.rootNode.addChildNode(node)
```

## 16.15 パスからの図形（**SCNShape**）

SCNText はフォントの情報からジオメトリを作成しますが、こちらは UIBezierPath でジオメトリを作成します。

設定値は SCNText 同様にイニシャライズや個別でパスを設定して、押し出し設定を行い、面取り (chamfer) も SCNText と同様の設定を行うことができます。

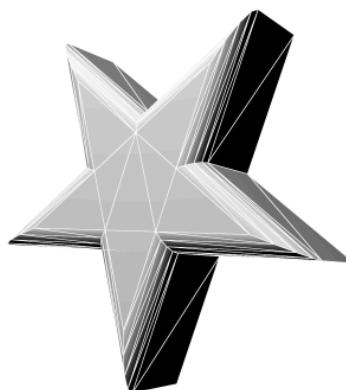


図 16.15 SCNSarpe

### **path**

描画する図形の UIBezierPath を設定します。

### **extrusionDepth**

押し出しの奥行きの値

### **chamferRadius**

面取りの半径を設定します。

### **chamferProfile**

面取りの形状 UIBezierPath を指定します

### **chamferMode**

面取りをする面の設定をします。 front は前面のみ、 back は後面のみ、 both 両面で初期値はこれが設定されている。

#### リスト 16.12: SCNText

```
// SCNShape
let shape = SCNShape()
```

```
let path = UIBezierPath()
path.move(to: CGPoint(x: 0, y: 0))
path.addLine(to: CGPoint(x: 0.2361, y: 0.7265))
path.addLine(to: CGPoint(x: 1.0, y: 0.7265))
path.addLine(to: CGPoint(x: 0.382, y: 1.1756))
path.addLine(to: CGPoint(x: 0.618, y: 1.9021))
path.addLine(to: CGPoint(x: 0.0, y: 1.4531))
path.addLine(to: CGPoint(x: -0.618, y: 1.9021))
path.addLine(to: CGPoint(x: -0.382, y: 1.1756))
path.addLine(to: CGPoint(x: -1.0, y: 0.7265))
path.addLine(to: CGPoint(x: -0.2361, y: 0.7265))

shape.path = path
shape.extrusionDepth = 0.5

shape.chamferRadius = 0.1
shape.chamferMode = .front

let node = SCNNNode(geometry: shape)
node.position = SCNVector3Zero
scene.rootNode.addChildNode(node)
```

# 第17章

## カスタムジオメトリ

---

この章では平面三角形のジオメトリをコードから作成します。わりと面倒なので知る必要がなければ、この章を読まず次章に移ってもらって構いません。

### 17.1 概要

ジオメトリの外観を構成するためには、以下の操作をする必要があります。

- ポリゴンを構成するために必要な点の情報 (Vertex)
- ポリゴン表示するための面を構成するために必要なインデックス情報
- ライトを適用する際に必要な法線情報 (Normal)
- テクスチャを適応するために必要なテクスチャ座標情報 (UV 座標)

### 17.2 SceneKit でカスタムジオメトリを作成する流れ。

- SCNGeometrySource でポリゴンの情報、法線情報、テクスチャ情報を設定
- SCNGeometryElement ポリゴンの面をどのように生成するか設定
- 2つを SCNGeometry に渡し、SCNNode の geometry で読み込みシーンに配置

今回はポリゴンの情報、法線情報、テクスチャ情報の設定を順々に設定していきます。

参考のため、カスタムジオメトリで使用している画像は iOS 10 のものです。iOS 11 で実行した際は法線の設定がない場合でも、法線が（自動？）設定されているようになってしましました。

## 17.3 シーンを作成する

UIViewController の viewDidLoad() に以下に設定。これ以降のコードはライトの以降に書いていきます。

リスト 17.1: シーンの初期設定

```
// シーン設定
let scnView = self.view as! SCNView
scnView.showsStatistics = true
scnView.allowsCameraControl = true

let scene = SCNScene()
scene.background.contents = UIColor.darkGray
scnView.scene = scene

// カメラ設定
let cameraNode = SCNNNode()
cameraNode.camera = SCNCamera()
cameraNode.position = SCNVector3(2.372, 1.473, 2.399)
cameraNode.eulerAngles = SCNVector3(-Float.pi * 0.125, Float.pi * 0.25, 0)
scene.rootNode.addChildNode(cameraNode)

// ライト設定
let lightNode = SCNNNode()
let omniLight = SCNLight()
omniLight.type = .omni
lightNode.light = omniLight
lightNode.position = SCNVector3(0, 5, 5)
scene.rootNode.addChildNode(lightNode)
```

## 17.4 三角形の平面を描画する

以下のコードをライトの設定の後に設定します。

リスト 17.2: 三角形の平面を描画

```
// SCNGeometry
let verticesArray = [
    SCNVector3(0, 1, 0),
    SCNVector3(-0.866, -0.5, 0),
    SCNVector3(0.866, -0.5, 0)
]

let indicesArray: [Int32] = [
    0, 1, 2
]

let vertices = SCNGeometrySource(vertices: verticesArray)
let faces = SCNGeometryElement(indices: indicesArray, primitiveType: .triangles)
let geometry = SCNGeometry(sources: [vertices], elements: [faces])

// SCNNNode
let node = SCNNNode(geometry: geometry)
scene.rootNode.addChildNode(node)
```



図 17.1 三角形の平面を描画

verticesArray の配列の中身は SCNVector3 で頂点の座標を 3 つ入れ正三角形を設定し

ます。

頂点情報だけですと何も描画されないため `indicesArray` で `verticesArray` に対応した面を描画するための数値を入れます。頂点の番号を取るだけなので整数の `Int32` です。

そして、頂点と面を設定します。`SCNGeometrySource` で頂点の配列とその数。`SCNGeometryElement` に面のインデックスの配列と面を貼っていきます。今回は、三角形で面を張っていますが、点、線、OpenGL などでもあった `Triangle Strip` などを使用することもできます。

`SCNGeometry` で頂点情報と面の情報を設定し、`SCNNode` の `geometry` へ設定します。

## 17.5 ライトの影響を受けるようにする

先ほど書いたものを消すか、変更部分を追加します。変更内容としては法線情報を追加しています。

リスト 17.3: 法線情報を追加

```
// SCNGeometry
let verticesArray = [
    SCNVector3(0, 1, 0),
    SCNVector3(-0.866, -0.5, 0),
    SCNVector3(0.866, -0.5, 0)
]

let indicesArray: [Int32] = [
    0, 1, 2
]

let normalsArray = [
    SCNVector3(0, 0, 1),
    SCNVector3(0, 0, 1),
    SCNVector3(0, 0, 1)
]

let vertices = SCNGeometrySource(vertices: verticesArray)
let normals = SCNGeometrySource(normals: normalsArray)
let faces = SCNGeometryElement(indices: indicesArray, primitiveType: .triangles)
let geometry = SCNGeometry(sources: [vertices, normals], elements: [faces])

// SCNNODE
let node = SCNNNode(geometry: geometry)
```

```
scene.rootNode.addChildNode(node)
```

わかりづらいですが、ライトが適応されています。



図 17.2 法線情報を追加

`normalsArray` で法線情報を配列に入れます。今回は正面を向いているので Z 軸しかありません。

`SCNGeometrySource` に `normalsArray` の法線情報の配列とその数を入れて、`SCNGeometry` の `sources` に法線情報を追加します。

## 17.6 テクスチャを適応する

`firstMaterial` や `SCNMaterial` にテクスチャを適応させたいのですが、UV 情報 (Texture Coordinates) がないため正しく設定されません。

頂点に対応する 2 次元の座標を設定することでテクスチャを貼る座標を設定することができます。



図 17.3 今回設定するテクスチャ/tex.png

リスト 17.4: 法線情報を追加

```
// SCNGeometry
let verticesArray = [
    SCNVector3(0, 1, 0),
    SCNVector3(-0.866, -0.5, 0),
    SCNVector3(0.866, -0.5, 0)
]

let indicesArray: [Int32] = [
    0, 1, 2
]

let normalsArray: [SCNVector3] = [
    SCNVector3(0, 0, 1),
    SCNVector3(0, 0, 1),
    SCNVector3(0, 0, 1)
]

let texcoordsArray: [CGPoint] = [
    CGPoint(x:0.5, y:0),
    CGPoint(x:0, y:1),
    CGPoint(x:1, y:1),
]
```

```
let vertices = SCNGeometrySource(vertices: verticesArray)
let normals = SCNGeometrySource(normals: normalsArray)
let texcoords = SCNGeometrySource(textureCoordinates: texcoordsArray)
let faces = SCNGeometryElement(indices: indicesArray, primitiveType: .triangles)
let geometry = SCNGeometry(sources: [vertices, normals, texcoords],
                           elements: [faces])

geometry.firstMaterial?.diffuse.contents = UIImage(named: "tex.png")
geometry.firstMaterial?.isDoubleSided = true

// SCNNNode
let node = SCNNNode(geometry: geometry)
scene.rootNode.addChildNode(node)
```

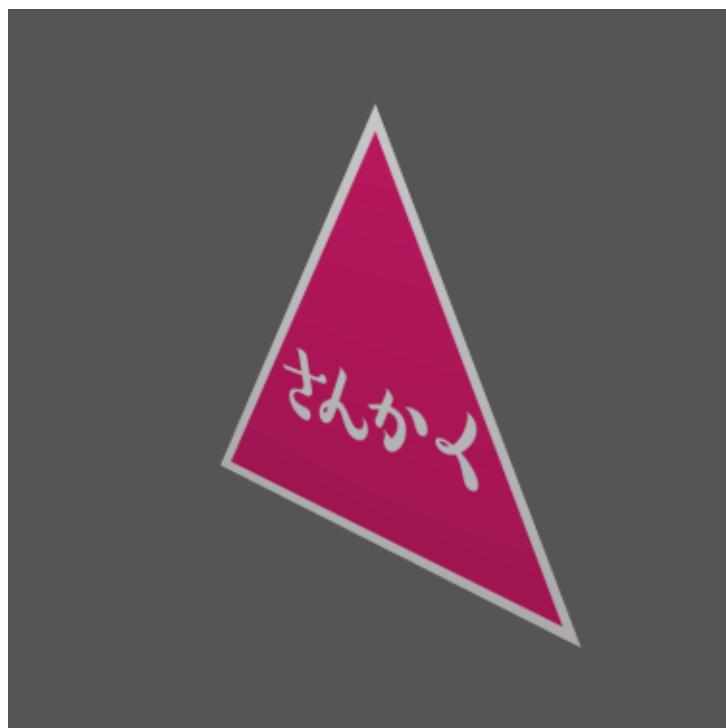


図 17.4 ジオメトリにテクスチャを適応

texcoordsArray 2 次元座標を CGPoint で設定し、SCNGeometrySource に textureCoordinates にこれを設定します。

そして、SCNGeometry の sources に追加し、firstMaterial の diffuse に設定。一応、裏面にライトの影ができるように isDoubleSided で両面表示する設定にしています。

# 第 18 章

## オブジェクトデータをシーンで使用する

SceneKit でのリソースの操作やオブジェクトファイルの読み込みは Model I/O のフレームワークを使用しております。

ファイルの読み込みは Model I/O に依存していますが、3DCG でメジャーなファイルフォーマットに対応しているため、大概のソフトから書き出したデータを使用できると思われます。

### 18.1 読み込むが可能なオブジェクトファイル

- COLLADA (.dae)
- Alembic (.abc)
- PLY ファイルフォーマット
- Standard Triangulated Language (.stl)
- Wavefront OBJ (.obj)
- Pixar Universal Scene Description (.usd/.usda)

ファイルフォーマット	ジオメトリ	マテリアル	アニメーション
DAE	○	○	○
ABC	○	×	○
PLY	○	×	×
STL	○	×	×
OBJ	○	○	×
USD	○	?	○

### 18.2 オブジェクトファイルの適応方法

そのままファイルをシーンに適応できます。

リスト 18.1:

```
let scene = SCNScene(named: "monkey.dae")!
```

### 18.3 Blender から DAE ファイルを作成する

基本的には、Export で COLLADA (dae) 書き出すだけです。書き出しオプションは、モディファイアの適応など状況に合わせて設定してください。

強いて言うならば書き出しオプションの Triangulate (三角面化) のチェックを外した方がよい場合があります。

### 18.4 Blender の DAE エクスポートで使用可能な機能

- ジオメトリ
- マテリアル情報 / UV 情報
- ボーン (Blender でいう Amarcher)
- アニメーション (IK 不可)
- カメラ
- ライト

モディファイアによるデフォームや IK は使用できませんので、変形をフリーズしたり、キーフレームをフレームごとに設定してください。

パーティクルはエクスポートできません。

### 18.5 Blender の DAE エクスポートでの注意点

SceneKit は右手座標ですが、Blender は左手座標となっており、CAD/CAM のアプリケーションでも左手座標のなっているものがあります。そのまま使用すると向きがおかしいので事前に変換するか、シーン上でジオメトリを回転する必要があります。

## 18.6 Mac のターミナルから **scn** ファイルを作成する

今回紹介したファイルフォーマットは、全てターミナルから **scn** ファイルに変化することができます。

```
$ xcrun scntool --convert cube.usda --format scn --output cube.scn
```

# 第19章

## アニメーションについて

---

SceneKit のアニメーションには大きく分けて以下の 2 種類があります。

- 物理アニメーションやパーティクルなど SceneKit が自動計算する
- アプリ開発者が任意で動きを設定する

### 19.1 アニメーションの方法の種類

SceneKit でパーティクルや物理アニメーションを除くと主な方法は以下の 3 つになります。

- SCNAction
- Core Animation
- SCNTransaction

また、iOS 11 で新しいアニメーションを方法の SCNAnimation や SCNAnimationPlayer が追加されました。

### 19.2 3 つのアニメーションの違い

#### 19.2.1 Core Animation と SCNAction

チルドノードである SCNNode をアニメーションすることができ、SCNAction は Core Animation の簡易版です。

UI を操作して弾を撃つ動作やキャラクターを移動させるなど、ユーザーのインタラクションによるアニメーションに適しており、繰り返し設定があるため、固定している物体のアニメーションにも使用できます。

## 19.2.2 SCNTransaction

シーン内のオブジェクトを一斉にアニメーションさせることができ、ゲームオーバー時で画面を初期状態に戻すなどの全体の変更に適しています。

## 19.3 アニメーションの適応範囲とアニメーションの繰り返し処理

以下、適応範囲と繰り返し処理についての表となります。SCNTransaction はシーン全体に影響を与え繰り返しではなく、他の 2 つはノードに紐付きアニメーションの繰り返しを行うことができます。

クラス名	適応箇所	アニメーションの繰り返し
Core Animation	ノード	○
SCNAction	ノード	○
SCNTransaction	シーン	×

## 19.4 アニメーションの同時設定、優先度

確信はないのですが、調べたところ、同じノードに SCNAction と Core Animation を同時に適応した場合は、SCNAction が優先されるようです。また、SCNAction か Core Animation でアニメーション設定したノードに対して、SCNTransaction でアニメーションをかけると互いの動作がミックスされます。

# 第20章

## シーン全体にアニメーションを与える **SCNTransaction**

---

前章で紹介 SCNTransaction。シーン全体にアニメーションを与える場合やタップなど動作が決まっているものに適していると思われます。

### 20.1 SCNTransaction の仕組み

SCNTransaction.begin() から SCNTransaction.commit() の中でアニメーションさせたい値を設定し、現在の状態から、その値をアニメーション終了時とし SCNTransaction.animationDuration で設定した時間までアニメーションされます。

公式のドキュメントで Animatable と書かれているものに関してはアニメーションすることができます、ほとんどの設定値をアニメーションさせることができます。

### 20.2 SCNTransaction.completionBlock

commit() 前に completionBlock で命令をかくと、アニメーション終了時にその命令が実行されます。completionBlock の中で SCNTransaction を実行するとアニメーションを入れ子にすることができます。

### 20.3 サンプルコードみる

以下は、Xcode の Game テンプレートのタップ後に書かれているアニメーションです。動作は画面をタップしジオメトリが見つかったら、ジオメトリのマテリアルの emission を赤く変化させて光っているような演出を与えています。

リスト 20.1: タッチイベント内のコード

## 第 20 章 シーン全体にアニメーションを与える SCNTransaction

```
// result からタッチしたジオメトリを検出しマテリアルを取得
let material = result.node.geometry!.firstMaterial!

// SCNTransaction 開始
SCNTransaction.begin()

// 0.5 秒間アニメーション
SCNTransaction.animationDuration = 0.5

// アニメーション終了時の命令
SCNTransaction.completionBlock = {
    // 終了時の SCNTransaction 開始
    SCNTransaction.begin()

    // 0.5 秒間アニメーション
    SCNTransaction.animationDuration = 0.5

    // emission を黒に戻す
    material.emission.contents = UIColor.black

    // 終了時の SCNTransaction 終了
    SCNTransaction.commit()
}

// マテリアルの emission を赤に変更
material.emission.contents = UIColor.red

// SCNTransaction 終了
SCNTransaction.commit()
```

SCNTransaction を開始し、5 秒でアニメーションを終了する設定にして、終了時に同様の 5 秒のアニメーションをしてマテリアルの emission を黒に戻す処理を設定。その後、内側のブロックで emission を赤に変更するして終了。

プログラム的にはこのようになる

- 開始 > 秒数設定 > 終了後の設定を開始 > 秒数設定 > マテリアルの色を戻す > 終了後の設定を終了 > マテリアルの色を赤 > 終了

が実際は下のようになる。

- 開始 > 秒数設定 > マテリアルの色を赤 > 終了 > 終了後の設定を開始 > 秒数設定 > マテリアルの色を戻す > 終了後の設定を終了

### 20.4 SCNTransaction.animationTimingFunction

今回は使用しませんが、アニメーションが始まり終わるまでのタイミングを設定することができ、所謂、イージングと呼ばれるものです。

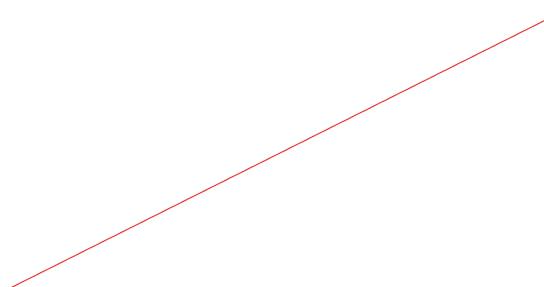
以下の設定か、カスタムでのタイミングを設定することができます。

- kCAMediaTimingFunctionLinear
- kCAMediaTimingFunctionEaseIn
- kCAMediaTimingFunctionEaseOut
- kCAMediaTimingFunctionEaseInEaseOut
- kCAMediaTimingFunctionDefault

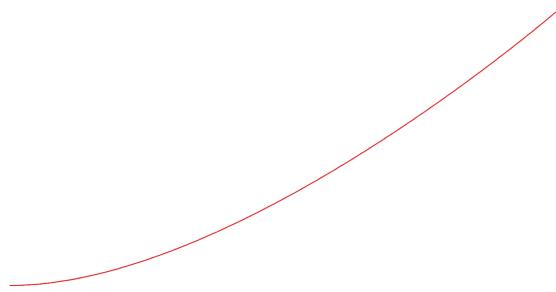
### 20.5 CAMediaTimingFunction 動作例

横が時間、縦がアニメーションの変化量です。また、アニメーションのタイミングをカスタマイズすることもできます。

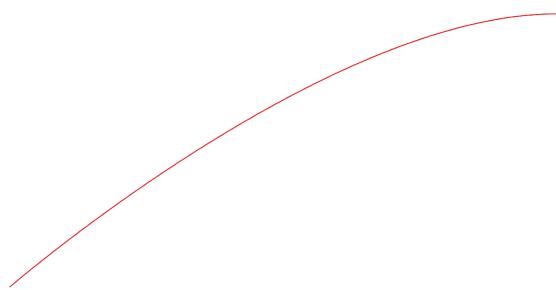
#### 20.5.1 Linear



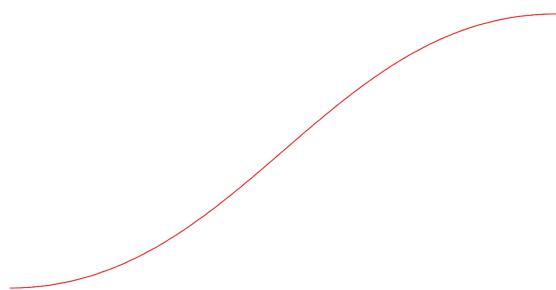
### 20.5.2 EaseIn



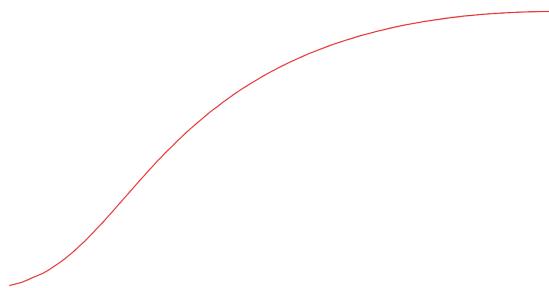
### 20.5.3 EaseOut



### 20.5.4 EaseInEaseOut



### 20.5.5 Default



## 20.6 その他

全てを適応してしまう flush や他のスレッドからのアニメーション変更から守る lock、 unlock、 disableActions などがあります。

# 第 21 章

## Core Animation を使用したアニメーション

iOS のアプリで使われる Core Animation の CABasicAnimation を使用し、Key を指定してアニメーションをさせます。

### 21.1 流れ

CABasicAnimation で SCNNode の rotation のキーパスを取得し、fromValue で最初の状態、toValue でアニメーション終了時の値を設定。アニメーションの再生時間と繰り返しなどを設定します。

### 21.2 サンプル

リスト 21.1: X 軸方向 45 度回転

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNode(geometry: box)

let rotate = CABasicAnimation(keyPath: "rotation")
rotate.fromValue    = SCNVector4(0.0, 0.0, 0.0, 0.0)
rotate.toValue     = SCNVector4(1.0, 0.0, 0.0, Float.pi * 2.0)
rotate.duration   = 10
rotate.repeatCount = .infinity // HUGE でも可
boxNode.addAnimation(rotate, forKey: "rotate")

scene.rootNode.addChildNode(boxNode)
```

四角のジオメトリを 10 秒で X 軸方向に 45 度回転し、「repeatCount = .infinity」にしてい

るため無限に設定。

`addAnimation` でノードに作成した「rotate」のアニメーションを追加しています。また、`addAnimation` の `forKey` は識別用の名前であるため、`nil` など空に設定しても動作します。

### 21.3 Core Animation の始まりと終わりを調べる

`CAAnimationDelegate` を使用すると `animationDidStart` や `animationDidStop` の関数の通知がされるため、アニメーションの始まりと `CAAnimation` から `forKey` で設定し、名前を調べて処理することができます。

# 第22章

## SCNAction（アクション）を使用したアニメーション

---

SCNAction は Core Animation のように様々なアニメーション可能なパラメーターを使用したアニメーションを標準ですることはできませんが、移動、回転、拡大縮小などを簡単なアニメーションをする場合は便利です。

また、カスタムのアクションの作成することで SCNAction にないパラメーターをアニメーションさせることができます。

### 22.1 SCNAction とは

SceneKit 用のアニメーションライブラリで、SCNAction で設定し、SCNNode でそのアニメーション設定を実行します。

以下、SCNAction 設定可能な振る舞いが可能です。

- 移動
- 回転
- 拡大
- フェードイン / フェードアウト
- 表示 / 非表示
- 一時停止
- 逆再生
- ノードの削除
- アニメーションのグループ化
- アニメーションを順番に再生する
- アニメーションのリピート

- カスタムアニメーションの設定
- 効果音など音の再生

### 22.2 アニメーションの再生方法

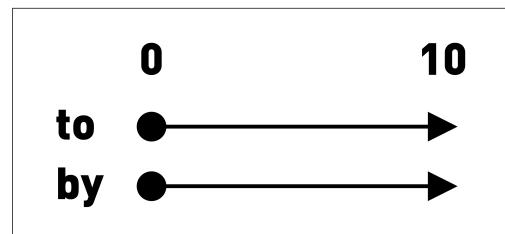
ノードが持つメソッド `runAction` で設定したアクションを呼ぶと、即アニメーションされます。そのため、使い勝手のよいアクションを設定しておくとアニメーションの動作を使い回しができます。

### 22.3 移動

目標座標まで進む `to` と 現在位置から移動を決める `by` のメソッドがあります。`to` は移動させる場所が決まっているもの、`by` は UI などからキャラクターなどを移動させるものに適しています。

`to` と `by` で 10 を設定した場合で座標上 0 を起点でアニメーションを始めると、終了時 10 を同じ座標値になりますが、5 を起点にして始めると、`to` は指定された値の 10 で止まり、`by` は現在の座標から 10 移動するため、終了する地点が変わります。

### to と by で 0 から 10 まで動かした場合



### to と by で 5 から 10 まで動かした場合

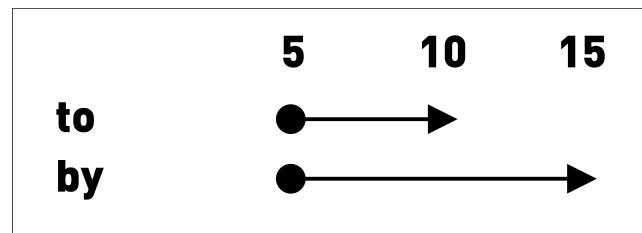


図 22.1 to と by での移動の違い

### 22.3.1 SCNAction.moveBy - x, y, z で指定

x に 2 を入れ X 軸方向に プラス 2 移動させており、duration はアニメーション終了するまでの秒数となります。

リスト 22.26:

```
let action = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(action)
```

by の値は移動量なのでもう一度行うと X 軸 4 の座標へ移動します。

リスト 22.26:

```
let action1 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(action1)
let action2 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(action2)
```

### 22.3.2 SCNAction.moveBy - SCNVector3 で指定

上と同じものを SCNVector3 で指定したものです。

リスト 22.26:

```
let action = SCNAction.move(by: SCNVector3(2, 0 ,0), duration: 2)
boxNode.runAction(action)
```

### 22.3.3 SCNAction.moveTo

以下のものを書くと X 座標 2 まで移動します。

リスト 22.26:

```
let action = SCNAction.move(to: SCNVector3(2, 0 ,0), duration: 2)
boxNode.runAction(action)
```

to は必ず、X 座標 2 へ移動させるため、by のように 2 回行っても 4 の座標には進みません。

リスト 22.26:

```
let action1 = SCNAction.move(to: SCNVector3(2, 0 ,0), duration: 2)
boxNode.runAction(action1)
let action2 = SCNAction.move(to: SCNVector3(2, 0 ,0), duration: 2)
boxNode.runAction(action2)
```

to で移動する座標はローカル座標です。そのため、SCNNode 入れ子にした場合、親の座標が X 軸 -1 にいて、子が X 方向 2 移動した場合、ワールド座標では X 方向 1 へ移動します。

## 22.4 回転

移動と同様に by と to のメソッドがあり、プラス値で反時計回りに回ります。

### 22.4.1 **SCNAction.rotateBy - x, y, z で指定**

以下、x, y, z に回転させたい角度をラジアンで入れて、duration で 2 秒を設定し、2 秒で Z 軸方向 45 度回転させています。今回は 角度 × (  $\pi / 180$  ) で角度を出しています。

リスト 22.26:

```
let action = SCNAction.rotateBy(x: 0, y: 0, z: CGFloat(45 * (Float.pi / 180)),  
duration: 2)  
boxNode.runAction(action)
```

### 22.4.2 **SCNAction.rotateTo - x, y, z で指定**

上の to のバージョン。移動同様に指定された角度に回転するため、45 度以上にはなりません。

リスト 22.26:

```
let action = SCNAction.rotateTo(x: 0, y: 0, z: CGFloat(45 * (Float.pi / 180)),  
duration: 2)  
boxNode.runAction(action)
```

### 22.4.3 **SCNAction.rotateBy - SCNVector3 で指定**

by の回転の SCNVector3 のバージョン。上のと同じように 2 秒で Z 軸方向 45 度回転させており、by で回転角度、around で回転する軸、duration で 2 秒を設定しています。

リスト 22.26:

```
let action = SCNAction.rotate(by: CGFloat(45 * (Float.pi / 180)),  
around: SCNVector3(0,0,1), duration: 2)  
boxNode.runAction(action)
```

### 22.4.4 SCNAction.rotateTo - SCNVector4 で指定

to の回転の SCNVector4 のバージョン。上のと同じように 2 秒で Z 軸方向 45 度に回転させており、toAxisAngle の SCNVector4 は 4 番目の w を前の 3 引数 x, y, z に適応させています。

リスト 22.26:

```
let action = SCNAction.rotate(toAxisAngle: SCNVector4(0, 0, 1,  
45 * (Float.pi / 180)), duration: 2)  
boxNode.runAction(action)
```

### 22.4.5 SCNAction.rotateTo - ShortestUnitArc を指定

このメソッドを使用すると現在地から最も近い回転位置で止まります。以下のコードでは 450 度、1 周と 90 度回転するはずですが、usesShortestUnitArc が true になっているため、最も近い位置の 90 度の位置で止まります。usesShortestUnitArc を false にするとデフォルト設定に戻るため 450 度回転します。

リスト 22.26:

```
let action2 = SCNAction.rotateTo(x: 0, y: 0, z: CGFloat(450 * (Float.pi / 180)),  
duration: 2, usesShortestUnitArc: true)  
boxNode.runAction(action2)
```

## 22.5 拡大縮小

移動と同様に `by` と `to` のメソッドがあり、`x, y, z` を 1 つの値で設定します。デフォルト値は 1 で、0 になると表示されなくなります。

マイナスを指定すると法線が逆になり、表示がおかしくなる可能性があるので注意が必要です。

### 22.5.1 **SCNAction.scaleBy**

現在の大きさから 2 秒で 2 倍の大きさにします。

リスト 22.26:

```
let action = SCNAction.scale(by: 2, duration: 2)
boxNode.runAction(action)
```

### 22.5.2 **SCNAction.scaleTo**

初期の大きさから 2 秒で 2 倍の大きさにします。ノードの `x, y, z` スケールのいずれかが `to` の値より大きい場合、アニメーションせずに即時でこの大きさになります。(バグ?)

リスト 22.26:

```
let action = SCNAction.scale(to: 2, duration: 2)
boxNode.runAction(action)
```

## 22.6 フェードイン / フェードアウト

完全に透明度が 0 や 1 になるメソッドと、透明度を指定するものがあり、移動と同様に透明度指定も `to` と `by` があります。

### 22.6.1 SCNAction.fadeOut - 徐々に消える

2 秒で透明度を 0 にして見えなくなります。

リスト 22.26:

```
let action = SCNAction.fadeOut(duration: 2)
boxNode.runAction(action)
```

### 22.6.2 SCNAction.fadeIn - 徐々に表示させる

2 秒で透明度を 1 にして表示します。ノードの透明度は 1 なので、最初に opacity で透明度を 0 になります。

リスト 22.26:

```
boxNode.opacity = 0
let action = SCNAction.fadeIn(duration: 2)
boxNode.runAction(action)
```

### 22.6.3 SCNAction.fadeOpacity - by

現在の透明度の値から 0.8 引いて 20 % の透明度になります。

リスト 22.26:

```
let action = SCNAction.fadeOpacity(by: -0.8, duration: 2)
boxNode.runAction(action)
```

### 22.6.4 SCNAction.fadeOpacity - to

現在の透明度の値から 20 % の透明度になります。

リスト 22.26:

```
let action = SCNAction.fadeOpacity(to: 0.2, duration: 2)
boxNode.runAction(action)
```

## 22.7 表示 / 非表示

ノードを表示 / 非表示し、内部的には SCNNODE の isHidden が false か true に設定されます。

リスト 22.26:

```
let action = SCNAction.hide()
boxNode.runAction(action)
```

リスト 22.26:

```
let action = SCNAction.unhide()
boxNode.runAction(action)
```

## 22.8 逆再生

逆再生と書いていますが、ドキュメントでは reversed() は戻す値で実行されると書かれています。そのため、以下のコードを入れると -X 軸方向に移動します。

後ほど説明するシーケンスのメソッドで使用すると簡単に逆再生ループの動作が作成できます。

リスト 22.26:

```
let action = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(action.reversed())
```

## 22.9 一時停止

アニメーションを一時停止させるために使用します。以下のコードでは、`runAction` の `completionHandler` を使用していますが、アニメーションだけなら、後ほど説明するシーケンスで使用するケースが多いと思われます。

### 22.9.1 一時停止する

2 秒停止して、その後 X 軸方向に 2 移動します。

リスト 22.26:

```
let action1 = SCNAction.wait(duration: 2)
let action2 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(action1, completionHandler: {
    boxNode.runAction(action2)
})
```

### 22.9.2 一時停止の振り幅を決める

`withRange` を設定することで、一時停止の時間に揺らぎを持たせることができます。

以下のコードでは、待ち時間 2 秒かつ ± 1 秒ランダムの終了時間となり、1 秒～3 秒の待ち時間となります。

リスト 22.26:

```
let action1 = SCNAction.wait(duration: 2, withRange: 2)
let action2 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(action1, completionHandler: {
    boxNode.runAction(action2)
})
```

```
})
```

### 22.10 アニメーションのグループ化

SCNAction をグループ化し、複数の SCNAction を同時に再生します。 SCNAction.group メソッドの配列をいくつかの SCNAction を入れるだけです。

以下のコードでは拡大と移動が同時に行われます。

リスト 22.26:

```
let action1 = SCNAction.scale(by:2, duration: 2)
let action2 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
boxNode.runAction(
    SCNAction.group([
        action1,
        action2
    ])
)
```

### 22.11 アニメーションを順番に再生する

複数の SCNAction を順番に再生します。 SCNAction.sequence メソッドの配列にいくつかの SCNAction を入れるだけです。

以下のコードでは、 X 軸方向に 2 移動し、 2 秒待機、 reversed メソッドで元に位置に戻ります。

リスト 22.26:

```
let action1 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
let action2 = SCNAction.wait(duration: 2)
boxNode.runAction(
    SCNAction.sequence([
        action1,
        action2,
    ])
)
```

```
        action1.reversed()
    ])
)
```

## 22.12 アニメーションのリピート

指定回数分リピートするアニメーションと、無限にリピートするアニメーションする設定があります。

### 22.12.1 指定した回数だけリピートする

上のシーケンスを使用したアニメーションを SCNAction.repeat で囲み、count を 4 に設定し 4 回繰り返します。

リスト 22.26:

```
let action1 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
let action2 = SCNAction.wait(duration: 2)
boxNode.runAction(
    SCNAction.repeat(
        SCNAction.sequence([
            action1,
            action2,
            action1.reversed()
        ]),
        count: 4
    )
)
```

### 22.12.2 無限にリピートする

repeatForever を使って先ほどの sequence を無限にリピートします。

リスト 22.26:

```
let action1 = SCNAction.moveBy(x: 2, y: 0, z: 0, duration: 2)
let action2 = SCNAction.wait(duration: 0.5)
boxNode.runAction(
    SCNAction.repeatForever(
        SCNAction.sequence([
            action1,
            action2,
            action1.reversed()
        ])
    )
)
```

### 22.13 ノードの削除

SCNAction に紐づいている SCNNNode をシーンから消すことができます。以下のコードでは scale で 4 倍の大きさにして、アニメーション終了後にシーンから削除されます。

リスト 22.26:

```
let action = SCNAction.scale(by: 4, duration: 2)
boxNode.runAction(
    SCNAction.sequence([
        action,
        SCNAction.removeFromParentNode()
    ])
)
```

### 22.14 timingMode と timingFunction の設定

SCNTransaction で使用していた CAMediaTimingFunction のような動作を使用することができます。

リスト 22.28: timingMode

```
let move = SCNAction.moveBy(x: 0, y: 2, z: 0, duration: 2)
move.timingMode = .easeInEaseOut;

boxNode.runAction(SCNAction.repeatForever(move))
```

`timingFunction` を使用して自分でアニメーションのタイミングを作ることもできます。以下のコードでは、経過時間を取り得し急速に Y 軸方向に移動します。

リスト 22.28: timingFunction

```
let move = SCNAction.moveBy(x: 0, y: 2, z: 0, duration: 2)

move.timingFunction = { (time:Float) -> Float in
    return time * time
}

boxNode.runAction(SCNAction.repeatForever(move))
```

`timingFunction` へ 0 を返し続けるとアプリの動作がおかしくなる可能性があるため、`timingFunction` のブロック内の記述は注意してください。

# 第23章

## SCNAction でカスタムアニメーションをつくる

---

SCNAction では customAction という命令があり、そのブロック内でアニメーションを設定することでノードの runAction で動作するものとなり、自前のアクションを作成することができます。

### 23.1 customAction の内容

customAction はドキュメントでこのようになっています。

リスト 23.1: customAction の内容

```
class func customAction(duration seconds: TimeInterval,  
                        action block: @escaping (SCNNode, CGFloat) -> Void) -> SCNAction
```

値は以下のような形

- seconds はこのアクションの再生時間
- block の引数 SCNNode は runAction で指定した時のノード
- block の引数 CGFloat はアニメーションの経過時間
- そして、ブロックの中がフレーム毎で動くため、受け取る SCNNode、CGFloat を使ってアニメーションをさせる

わかりづらいので、下のコードで見ていきましょう。

## 23.2 コードサンプル

以下のコードは四角のジオメトリを黒から赤に点滅させるものです。

`duration` の定数でアニメーション時間を設定し、`sampleAction` で設定している `customAction` に先ほど設定した `duration` を設定します。

`action` のブロックはフレーム毎に呼ばれ、1つ目の引数 (`node`) で `runAction` で設定したノード、2つ目の引数 (`elapsedTime`) で経過時間が取得できるため、これらを使用してアニメーションを作ります。

経過時間をアニメーション時間で割ると、フレーム毎で呼び出された時に状態が 0 から 1 の間となるので、この値を使用して色を変えていきます。

リスト 23.2: 黒から赤に点滅させる

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

scene.rootNode.addChildNode(boxNode)

// カスタムアクションを設定
// 黒から赤に点滅
let duration:TimeInterval = 1.0
let sampleAction = SCNAction.customAction(duration: duration,
action: { (node, elapsedTime) in
    let percentage = elapsedTime / CGFloat(duration)
    node.geometry?.firstMaterial?.diffuse.contents =
    UIColor.init(red: percentage, green: 0, blue: 0, alpha: 1.0)
})

// カスタムアクションを repeatForever の無限ループで再生
boxNode.runAction(SCNAction.repeatForever(sampleAction))
```

## 23.3 注意点

他のアクションもそうなのですが、アクション実行されると SceneKit はアクションのアニメーションが終了するまでブロックを繰り返し呼び出し、経過時間を計算し、それをブロック

に渡します。

ブロック内の再生は 1 度再生を始めると終わるまで再生し、この操作を元に戻すことができません。要するにブロックで再生されたら、ノードを消したり、`isPaused` で一時停止しない限り、終わるまで再生されてしまいます。

# 第24章

## モーフアニメーション

---

SceneKit では SCNNode に SCNMorpher が紐づけられており、1つまたは複数のジオメトリを配列として渡すと、各ジオメトリへ変形させることができるモーフィングが使用できます。

3DCG のオーサリングツールでは Blend Shape と呼ばれているものもあり、主にキャラクターの表情などで使われます。

A のジオメトリから B のジオメトリへポリゴンの頂点が変形させられるため、ジオメトリの頂点の数が一致していないとアニメーションできず、頂点の順番がおかしいとアニメーション時にジオメトリの面がおかしくなるので注意が必要です。

### 24.1 SCNMorpher の振る舞い

A のジオメトリから B のジオメトリ変形させる際、ジオメトリの配列の要素を指定します。そして、weight という変化の比率の値を変更するとジオメトリが変形します。

現状 SCNMorpher は以下のものがあります。

- targets
- weight
- setWeight
- calculationMode
- unifiesNormals

targets に変化させる SCNGeometry が配列になっているものを入れて、setWeight で配列の番号と、変化させる度合いを数値で入れます。weight は配列の番号で値の状態を調べます。

calculationMode は Normalized と Additive があり、Normalized は変化量を 0 ~ 1 に正規化され、Additive 元の頂点の位置から加えて頂点が変形されます。初期値は Normalized。

```
let box1 = SCNBox(width: 1, height: 5, length: 1, chamferRadius: 0.01)
let box1Node = SCNNNode(geometry: box1)
box1Node.name = "box1"
scene.rootNode.addChildNode(box1Node)

let box2 = SCNBox(width: 5, height: 1, length: 1, chamferRadius: 0.5)

let morpher = SCNMorpher()
morpher.targets = [box2]
box1Node.morpher = morpher

let a2 = CABasicAnimation.init(keyPath: "morpher.weights[0]")
a2.fromValue = 0.0
a2.toValue = 1.0
a2.duration = 0.5
a2.beginTime = 1.0
a2.autoreverses = true
a2.repeatCount = .infinity // HUGE でも可

box1Node.addAnimation(a2, forKey: nil)
```

## 24.2 iOS 11 で追加された機能

ジオメトリで設定した name から weight の設定や取得ができるようになり、SCN-Morpher で設定された weights を使用すると、設定したウェイトの値をまとめて取得できるようになりました。

リスト 24.1: iOS 11 で追加された機能

```
let box1 = SCNBox(width: 1, height: 5, length: 1, chamferRadius: 0.01)
let box1Node = SCNNNode(geometry: box1)
scene.rootNode.addChildNode(box1Node)

let box2 = SCNBox(width: 5, height: 1, length: 1, chamferRadius: 0.5)
box2.name = "box2"

let box3 = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 1.0)
box3.name = "box3"
```

```
let morpher = SCNMorpher()
morpher.targets = [box2,box3]
box1Node.morpher = morpher

morpher.setWeight(0.2, forTargetNamed: "box2") // ジオメトリの名前で検索し設定
morpher.setWeight(1.0, forTargetNamed: "box3") // ジオメトリの名前で検索し設定

print(morpher.weights) // [0.2, 1.0] と weight の値が配列で返ってくる
```

また、もう1つ追加された `unifiesNormals` は `true` にすると、法線は変形されませんが、代わりに頂点をモーフィングした後に再計算されます。

### 24.3 Blender から書き出された `dae` ファイルを使用する

普通に Shape Key を作成して `dae` に書き出します。

そのままでは、`SCNMorpher` に変換されないので以下の作業を行います。

- `library_visual_scenes` の `instance_geometry` を `instance_controller` に変更する
- その上にある `controller` の名称を `instance_controller` に設定する

例えば、こちらを

```
<controller id="o-morph" name="o-morph"> . . .
```

こちらに変更します

```
<instance_controller name="o-morph" url="#o-morph"> . . .
```

また、Xcode でコンパイルする必要がありますが、SceneKit で読み込むモーフ設定にするツールを github で公開している方がおります。<sup>1</sup>

\*1 <https://github.com/JonAllee/ColladaMorphAdjuster>

## 24.4 Maya から書き出された dae ファイルを使用する

Blend Shape Deformer のものや、ベースオブジェクトだけでやっているものでも可能ですが、Maya から dae に書き出して、Geometry Morphers から数値を変えると、数値を変更した分だけジオメトリが大きくなってしまいます。

dae ファイル内の morph にある method が RELATIVE になっているので NORMALIZED に変更する必要があります。

ちなみに Maya LT からの変換はできません。Maya LT から dae のエクスポートできない仕様と、FBX で書き出して FBX Converter から dae に書き出そうとしてもモーフなど特定のものがある場合、書き出しエラーが起り 0 KByte のファイルができるためです。

### 24.4.1 サンプルコード

scn ファイルを読み込んでいるだけで、先程のコードとあまり変わりません。

```
let scene = SCNScene(named: "monkey.scn")!

let o1 = scene.rootNode.childNode(withName: "o", recursively: true)!

let a2 = CABasicAnimation.init(keyPath: "morpher.weights[0]")
a2.fromValue = 0.0
a2.toValue = 1.0
a2.duration = 0.5
a2.beginTime = 1.0
a2.autoreverses = true
a2.repeatCount = .infinity // HUGE でも可

o1.addAnimation(a2, forKey: nil)
```

以下の画像では、Blender の Suzanne を球体の形状にモーフィングさせています。

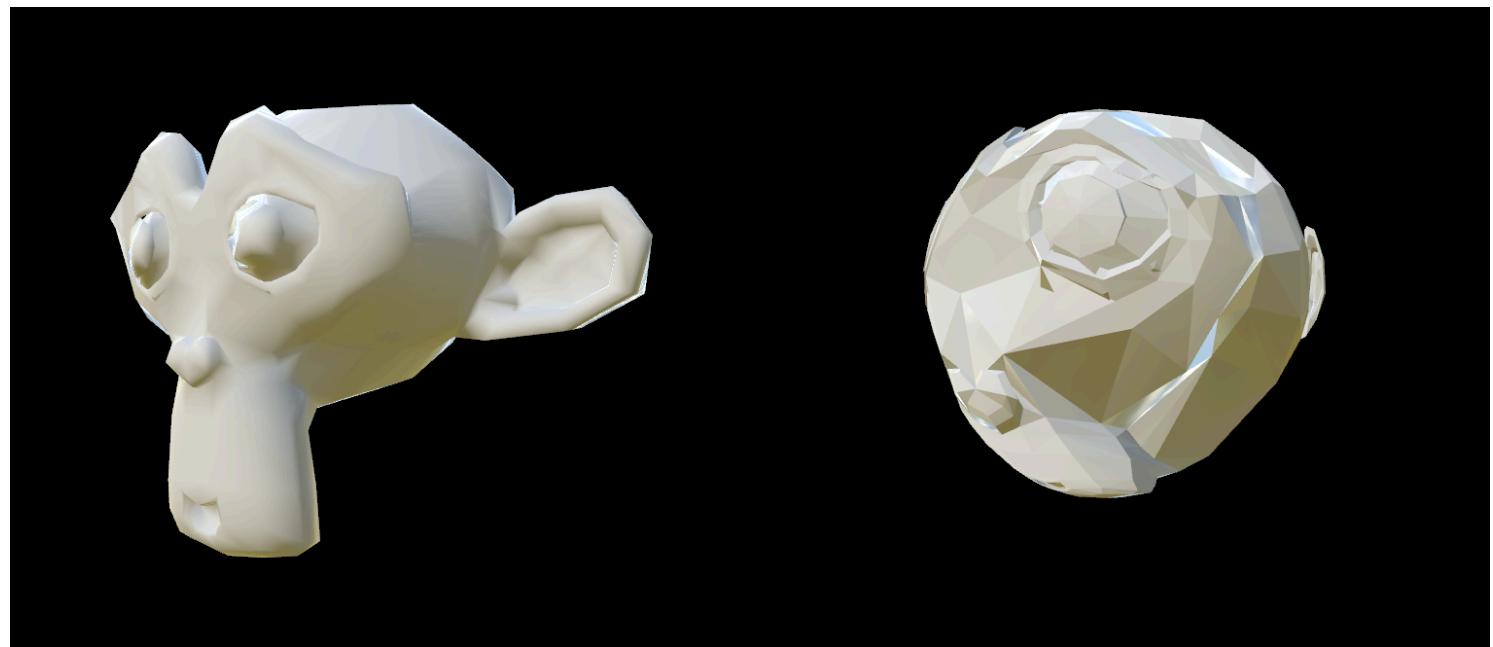


図 24.1 Blender から書き出したものを scn に変換したもの

## 24.5 注意点

同じ量の頂点が必要なため容量が大きくなる可能性があります。また、法線を元に各頂点を移動させるため処理も重い部類になります。

# 第 25 章

## アニメーション機能 **SCNAnimation** と **SCNAnimationPlayer**

---

アニメーションの呼び出し、再生、停止など簡単にする SCNAnimationPlayer とその設定値である SCNAnimation が iOS 11 で追加されました。

### 25.1 他のアニメーションとの違い

SCNAnimationPlayer の振る舞いはノード単位でつける Core Animation や Action と同じですが、最大の利点は Core Animation や Core Animation で簡単にできなかった再生スピードの伸縮や、複数のアニメーションの適応度合いを設定しブレンドができます。

### 25.2 今回の設定の流れ

- ・シーンファイル (.scn) やオブジェクトのファイルを読み込む
- ・enumerateChildNodes でアニメーションを調べ取り出し SCNAnimationPlayer に設定
- ・設定した SCNAnimationPlayer をアニメーションさせるノードの addAnimationPlayer を設定する

今回は SCNAnimationPlayer に設定する SCNAnimation を .scn から行なっていますが、アニメーションファイル (.scnanim) や Core Animation から設定することもできます。

### 25.3 設定例

タップするとアライグマが回転します。

わかりやすくするため enumerateChildNodes をファイルロードのたびに使用しています。

必要であれば関数などでまとめてください。

### リスト 25.1: SCNAnimationPlayer

```
import PlaygroundSupport
import SceneKit

class GameViewController: UIViewController {

    var scnView:SCNView!

    var Max:SCNNode!

    override func viewDidLoad() {
        super.viewDidLoad()

        // シーンファイルの呼び出し
        let scene = SCNScene(named: "max.scn")!

        // カメラ
        let cameraNode = SCNNode()
        cameraNode.camera = SCNCamera()
        cameraNode.position = SCNVector3(x: 0.636, y: 0.199, z: 1.374)
        cameraNode.eulerAngles = SCNVector3(x: 0.636, y: 0.125 * Float.pi, z: 0)
        scene.rootNode.addChildNode(cameraNode)

        // オムニライト
        let lightNode = SCNNode()
        lightNode.light = SCNLight()
        lightNode.light!.type = .omni
        lightNode.position = SCNVector3(x: 0, y: 10, z: 10)
        scene.rootNode.addChildNode(lightNode)

        // アンビエントライト
        let ambientLightNode = SCNNode()
        ambientLightNode.light = SCNLight()
        ambientLightNode.light!.type = .ambient
        ambientLightNode.light!.color = UIColor.darkGray
        scene.rootNode.addChildNode(ambientLightNode)

        // View 設定
        scnView = SCNView()
        self.view.addSubview(scnView)

        // View の Autolayout
```

```

scnView.translatesAutoresizingMaskIntoConstraints = false
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "V:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "H:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))


scnView.scene = scene
scnView.allowsCameraControl = true
scnView.showsStatistics = true
scnView.backgroundColor = UIColor.black


let walkScene = SCNScene(named: "max_walk.scn")!

var walkAnimation: SCNAnimationPlayer!

walkScene.rootNode.enumerateChildNodes { (child, stop) in
    if !child.animationKeys.isEmpty {
        walkAnimation = child.animationPlayer(forKey:
            child.animationKeys[0])
        stop.pointee = true
    }
}

walkAnimation.speed = 1

let spinScene = SCNScene(named: "max_spin.scn")!

var spinAnimation: SCNAnimationPlayer!

spinScene.rootNode.enumerateChildNodes { (child, stop) in
    if !child.animationKeys.isEmpty {
        spinAnimation = child.animationPlayer(forKey:
            child.animationKeys[0])
        stop.pointee = true
    }
}

spinAnimation.speed = 1.5
//spinAnimation.blendFactor = 0.5
//spinAnimation.animation.autoreverses = true
spinAnimation.animation.isRemovedOnCompletion = false
spinAnimation.stop()

```

```
Max = scene.rootNode.childNode( withName: "Max_rootNode",
recursively: true)!

Max.addAnimationPlayer(walkAnimation, forKey: "walk")
Max.addAnimationPlayer(spinAnimation, forKey: "spin")

// タップジェスチャー
let tapGesture = UITapGestureRecognizer(target: self, action:
    #selector(handleTap(_:)))
scnView.addGestureRecognizer(tapGesture)
}

@objc
func handleTap(_ gestureRecognize: UIGestureRecognizer) {
    Max.animationPlayer(forKey: "spin")?.play()
}

override var prefersStatusBarHidden: Bool {
    return true
}

override var supportedInterfaceOrientations: UIInterfaceOrientationMask {
    if UIDevice.current.userInterfaceIdiom == .phone {
        return .allButUpsideDown
    } else {
        return .all
    }
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
}

}

PlaygroundColor.current.liveView = GameViewController()
```

spinAnimation の speed を 0.2 などにして再生スピードを落としたり、spinAnimation.blendFactor のコメントアウトを外して、回転するアニメーションの適応度合いを半分にしてみたり、autoreverses のコメントアウトを外して逆再生をさせてもよいかと思い

ます。

## 25.4 SCNAnimation と SCNAnimationPlayer の設定値と関数

各プロパティで Bool 値のものは初期値が全て false になっています。

表 25.1 SCNAnimation

名称	説明
animationDidStart	アニメーション開始時にブロック内の命令を実行する
animationDidStop	アニメーション停止時にブロック内の命令を実行する
animationEvents	SCNAnimationEvent を設定する
autoreverses	再生後、逆再生を行い停止する。
blendInDuration	ブレンド開始時からブレンドするまでの時間を決める
blendOutDuration	ブレンド終了時から元に戻るまでの時間を決める
duration	アニメーション再生時間設定、初期値は 0 で設定されている時間を使う
fillsBackward	true の場合、アニメーションより前にアクティブになり処理がされる
fillsForward	true の場合、アニメーション後にアクティブになり処理がされる
isAdditive	アニメーションの値が追加される設定
isAppliedOnCompletion	true の場合、再生が終わるとアニメーションがモデルツリーに追加される
isCumulative	繰り返し時アニメーションの値が累積される設定
isRemovedOnCompletion	再生終了に SCNAnimation を破棄する。 (true の場合、再度設定しないと再生できません)
keyPath	呼び出し用のキーパス
repeatCount	リピート回数
startDelay	Core Animation の beginTime のブリッジ。遅らせて再生を開始する秒数
timeOffset	再生しているローカル時間からオフセットオフセットする時間
timingFunction	SCNTimingFunction を設定する
usesSceneTimeBase	初期値ではシステムの時間を使うが、シーンの時間を使う

表 25.2 SCNAnimationPlayer

名称	説明
animation	アニメーションさせる SCNAnimation を設定する
blendFactor	他のアニメーションのにどのくらいブレンドするかを 0~1 で設定する
paused	アニメーションの一時停止するか否か
speed	再生スピード。SCNAnimation の duration を元に再生時間を伸縮する
play()	SCNAnimation を再生する
stop()	SCNAnimation を停止する
stop( withBlendOutDuration: TimeInterval)	SCNAnimation を停止し、 その後他のアニメーションとブレンドさせながら戻す時間を決める

# 第26章

## マテリアル

---

ジオメトリの表面の見た目を決めるものがマテリアルとなり、SceneKit ではマテリアルの設定方法がいくつかあります。

### 26.1 その前にマテリアルは何をしているの？

ジオメトリの法線情報を元にジオメトリがどのような色の光を反射するかを決め、赤い物体は赤色を反射し、光が強いとハイライトが表示されます。

現実世界では、物体の表面自体がその色を持っているわけではなく、照らされた光を反射し、その色以外を吸収しているような振る舞いをしています。

### 26.2 ジオメトリへのマテリアルの適応方法

ジオメトリへのマテリアルの適応方法は以下の 3 つで、SCNGeometry には materials という項目があり、いくつかのマテリアルを配列に渡して適応させることができます。

- firstMaterial や SCNMaterial で用意されているプロパティを設定する
- シェーダーのスニペットをテキストで書いて設定する
- SCNProgram から Metal や GLSL 使用したカスタムシェーダーを書く

### 26.3 SCNMaterial の基本設定

SCNMaterial の主な項目が SCNMaterialProperty で設定されており、そこにはない機能と合わさったものが SCNMaterial の基本的な機能になっています。

SCNMaterial で SCNMaterialProperty を除いた場合、主な機能は以下の 4 つ

- ライティングモデルの設定、

- 透過や光沢の設定（光沢は一部ライトモデルのみ使用可能）
- ポリゴンの表裏、深度情報、色などから行う表示設定。
- プログラム上から調べる際に使用するマテリアルの名前の設定

また、シェーダーのスニペットを適応する `shaderModifiers` と、カスタムシェーダーを適応する `program` があります。シェーダースニペットやカスタムシェーダーなどのプログラム上から変更するものについては、それだけで分厚い本ができてしまうので今回は割愛させていただきます。

## 26.4 ライティングモデルとは？

ジオメトリの表面を設定する際に、光を当たり方をどのような方法を使用して設定するかという値です。主な設定は以下のものです。

- Constant
- Lambert
- Blinn
- Phong
- Physically Based

本書では Constant と Physically Based しか使っていないため、Physically Based の説明に重点をおきます。また、実際マテリアルを使用する場合も Physically Based がほとんどとなると思います。

### 26.4.1 各ライティングモデルの説明

#### Constant

単色で塗られ光の影響を受けない質感。

#### Lambert

光沢のないつや消しを行なっている物体に向いている質感。

ハイライトはつきません。

### Blinn

金属のような広めのハイライトをもつ物体に向いている質感。

### Phong

鏡面反射を行うような物体のような強い光沢を持つ質感。Normal マップでは、ロジック的な問題で状況によっては光沢部分が汚くなる可能性があります。

### Physically Based (要 Metal)

Lambert、Blinn、Phong は計算で近似した質感を計算を行いますが、Physically Based は光学的に現実世界に近いリアルなジオメトリの表示を行います。

また、Metal で実行していないと Physically Based を使用することはできません。OpenGL で実行すると真っ黒で表示されます。

## 26.5 ライティングモデルでの負荷

Constant、Lambert、Blinn、Phong、Blinn、Physically Based の順で処理が重くなりますが、いくつかの機能が Physically Based でしか動作しないため、基本的には Physically Based の設定で動作させることになります。

## 26.6 マテリアルの基本設定

### 26.6.1 透明度と値とモード

透過を設定する際、透過する値と透過するモードを設定します

#### 値

0 から 1 となりの 0 にすると透明になりますが、rgbZero は逆になり 1 が透明の設定になります。

#### モード

以下のモードがあります。

- default (旧 aOne)
- rgbZero
- singleLayer
- dualLayer

### default

通常の透過となり、自身のジオメトリ含むんだ奥のものが透過されます。

### rgbZero

色の輝度から透明度を返します。

### singleLayer

default と同じような透過ですが、自身のジオメトリが手前にあっても透過されません。

### dualLayer

dualLayer を設定すると、自身のポリゴンの裏面も透過の対象になります。X 線写真のような透過になります。

#### リスト 26.1: 透明度とモード

```
boxNode.geometry?.firstMaterial?.transparency = 0.5  
boxNode.geometry?.firstMaterial?.transparencyMode = .default
```

## 26.6.2 Double sided

true でジオメトリのポリゴンの裏面も描画処理し、主に厚みのない平面の板などで使用します。

#### リスト 26.2: Double sided

```
boxNode.geometry?.firstMaterial?.isDoubleSided = true
```

### 26.6.3 Lit per pixel

ジオメトリのレンダリング時にピクセル毎でライティングの計算を実行します。

チェックを外すと頂点からライティングの計算を行い補完した値でジオメトリのマテリアルを表示することで、見た目にはほとんど影響を与えずにレンダリングのパフォーマンスを向上させることができます。

リスト 26.3: Lit per pixel

```
boxNode.geometry?.firstMaterial?.isLitPerPixel = true
```

### 26.6.4 Cull Mode

ジオメトリの面が向きを設定し、設定を変更すると法線情報を反転させて逆側を表示することができます。

デフォルトでは Cull front が設定されており、Cull back にすると反転され裏面が表示されます。

リスト 26.4: cullMode

```
boxNode.geometry?.firstMaterial?.cullMode = .front
```

### 26.6.5 Blend Mode

画面上でのジオメトリの描画モードを設定します。初期値のジオメトリのアルファ値でブレンド Alpha を除くと以下の 6 つになります。

ブレンドモード	説明
Add	背景色からジオメトリの色を加算してブレンドする
Subtract	背景色からジオメトリの色を減算してブレンドする
Multiply	ジオメトリの色と背景色を掛け合わせてブレンドする
Screen	ジオメトリの色の逆数に背景色の逆数を掛けけてブレンドする
Replace	背景色をジオメトリの色に置き換え、アルファを無視してブレンドする (Alpha と同じだが透明にならない)
Max	ジオメトリの色と背景どちらかの大きい値を取り、 アルファを無視してブレンドする。iOS 11 で追加



図 26.1 ブレンドモード

上段 : Alpha 中段 : Add、Subtract、Multiply 下段 : Screen、Replace、Max

### 26.6.6 Write depth

false にすると、半透明オブジェクトをレンダリングするように深度情報の書き込みを無効にします。

リスト 26.5: writesToDepthBuffer

```
boxNode.geometry?.firstMaterial?.writesToDepthBuffer = false
```

### 26.6.7 Reads depth

false にすると、レンダリング時に深度情報の読み込みを無効にして最前面に表示します。ゲームなどの UI や HUD などを表現するためものもだと思われます。

リスト 26.6: readsFromDepthBuffer

```
boxNode.geometry?.firstMaterial?.readsFromDepthBuffer = false
```

### 26.6.8 Lock ambient with diffuse

true にする、もしくは Physically Based の場合には、環境光 (Ambient) と表面 (Diffuse) が両方に同じように応答します。

そのため、チェック時は Ambient にテクスチャが適応されない。

リスト 26.7: locksAmbientWithDiffuse

```
boxNode.geometry?.firstMaterial?.locksAmbientWithDiffuse = false
```

# 第27章

## マテリアルプロパティ

---

SceneKit のマテリアルは複数のプロパティで構成されており、マテリアルプロパティの情報や、ライトの位置、強度、色など組み合わせ、シーン内をピクセルのデータとしてレンダリングします。

また、本書では Physically Based のライティングモデルの使用がほとんどですので、Physically Based で使用している項目のみご紹介します。

### 27.1 Physically Based が持つヴィジュアルプロパティ

3DCG やゲームエンジンなどでよくあるマテリアルの設定で SceneKit ではヴィジュアルプロパティと呼んでいます。<sup>\*1</sup>以下の各ヴィジュアルプロパティにはマテリアルプロパティの contents、intensity という設定値とテクスチャマッピングの際の設定、iOS 11 で追加されたテクスチャの色やアルファの適応設定などがあります。

- Diffuse
- Metalness
- Roughness
- Normal
- Occlusion
- Illumination
- Emission
- Displacement (iOS 11 で追加)

---

<sup>\*1</sup> ヴィジュアルプロパティをこの章で紹介していますが、機能的には SCNMaterial の機能です。

## 27.2 マテリアルプロパティの各パラメーターが持つ値

`firstMaterial` の `diffuse` などで、すでに使用していますが、これらのパラメーターは主に 2 つの値を持っています。表面を表す `contents` と その適応度合いを表す `intensity` です。

```
node.geometry?.firstMaterial?.contents = UIColor.red  
node.geometry?.firstMaterial?.intensity = 0.5
```

### 27.2.1 contents

マテリアルプロパティはジオメトリの表面全体に均一な色で彩色するカラーと画像を貼り付けて質感を表すテクスチャのいずれかを設定することができます。

またテクスチャをして使用できるものは以下のものになります。

- `UIImage` / `NSImage` で設定する画像ファイル
- 指定した大きさ、もしくは 6 枚の画像を使用したキューブマップ（要 Metal）
- Core Animation レイヤー（macOS のみ）
- SpriteKit のテクスチャ画像、または SpriteKit のシーン自体

### 27.2.2 intensity

`contents` で設定した色やテクスチャの適応度合いを表し、初期値は 1 となっています。

この値は物体に対して光から影響を受ける度合いで、0 が黒となり、値を上げることで黒から色の成分が足され 1 で指定した値になります。

1 以上にすることも可能ですが、色が白に近くなります。

## 27.3 ヴィジュアルプロパティ (Physically Based)

### 27.3.1 Diffuse

表面色の設定で、色を設定していても `intensity` が 0 ですと 黒になります。他の設定が反映されない可能性があるため、完全に色を反射するマテリアル以外は 0 の値をとる完全な黒にしない方がよいです。

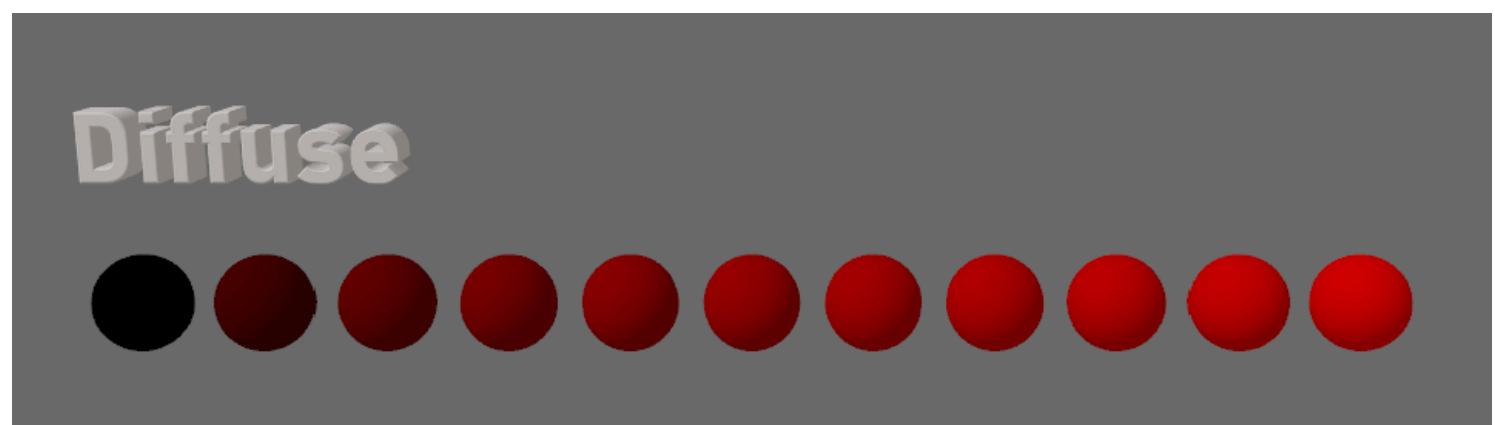


図 27.1 Diffuse

### 27.3.2 Metalness

金属や非金属を表面設定を行い、色またはグレイスケールの画像と `intensity` で適応具合を調整します。

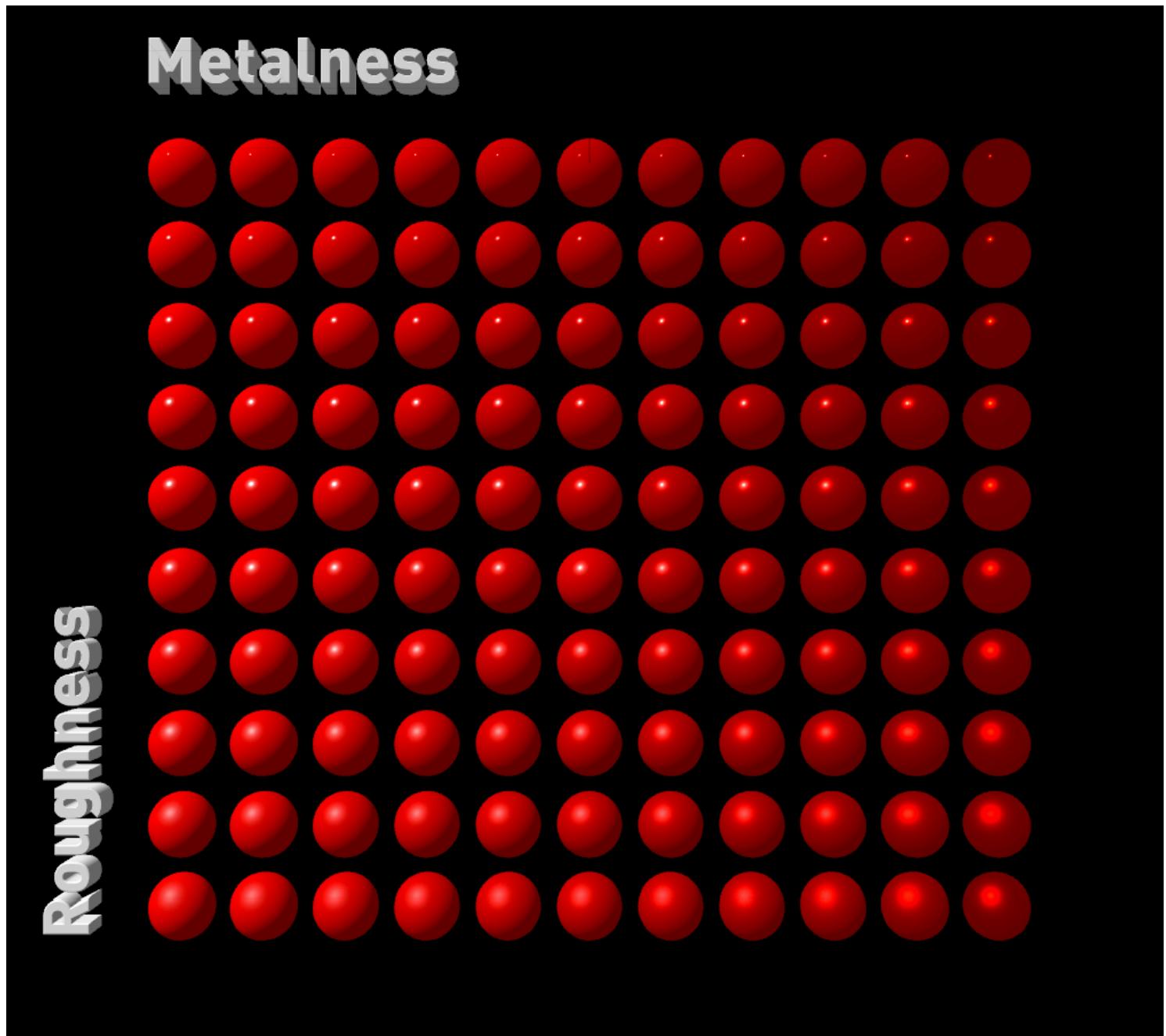


図 27.2 Metalness と Roughness を intensity で調整したもの

### 27.3.3 Roughness

オブジェクトの表面の細かな荒さを設定し、つや消しのような広がった光沢や金属のよう鋭い光沢の設定を行います。色またはグレイスケールの画像と intensity で適応具合を調整します。

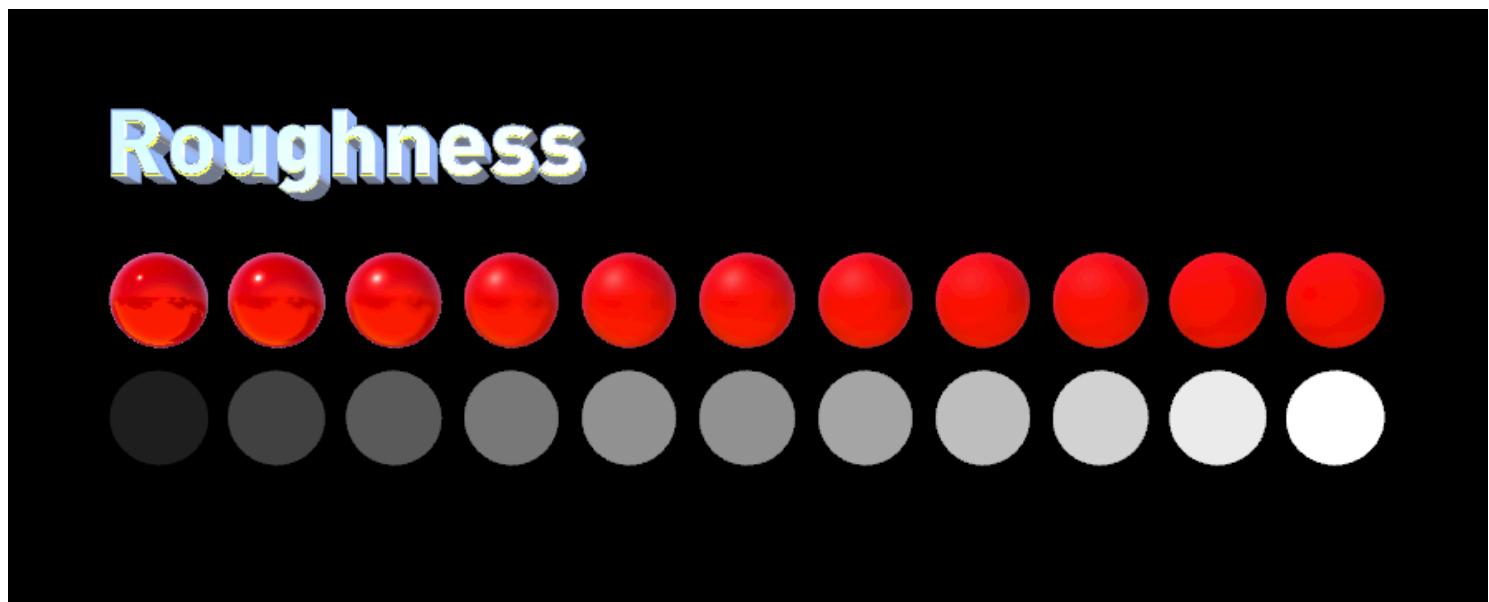


図 27.3 Roughness を色で調整したもの

#### 27.3.4 Normal

Normal マップ使用時の設定です。テクスチャ画像のみ使用可能で、わかりづらいのですが、画像の値から法線情報を変更し凹凸のあるような表面をつくります。Normal マップ用の画像が必要になるため、通常の画像を使用することはできません。

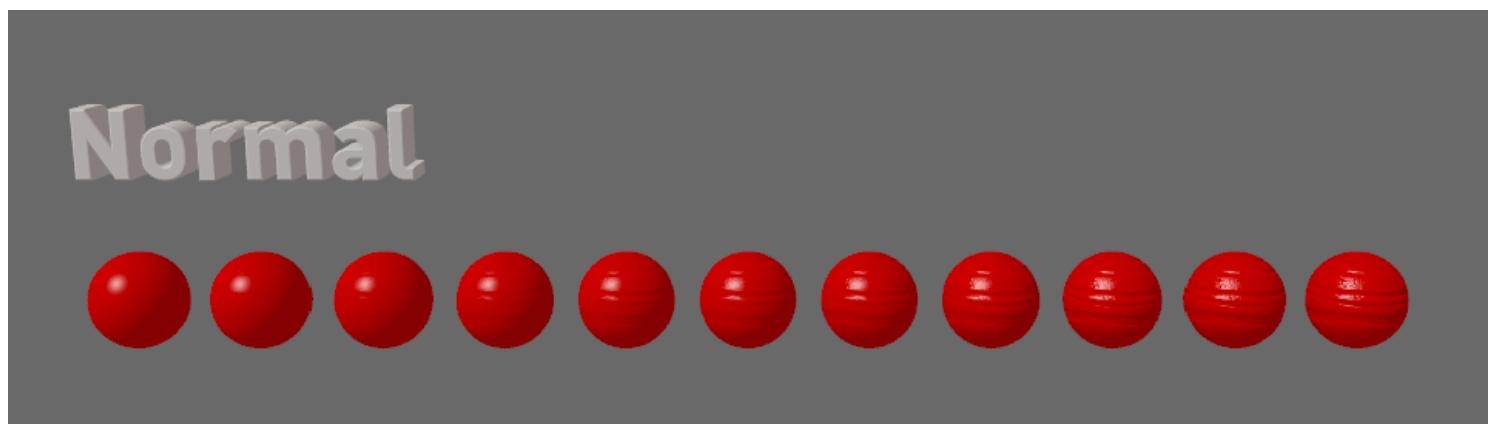


図 27.4 Normal

頂点や面を変更するわけではないので輪郭は元のジオメトリのそのままとなります。画像では輪郭に隆起ができるはずですが真円です。

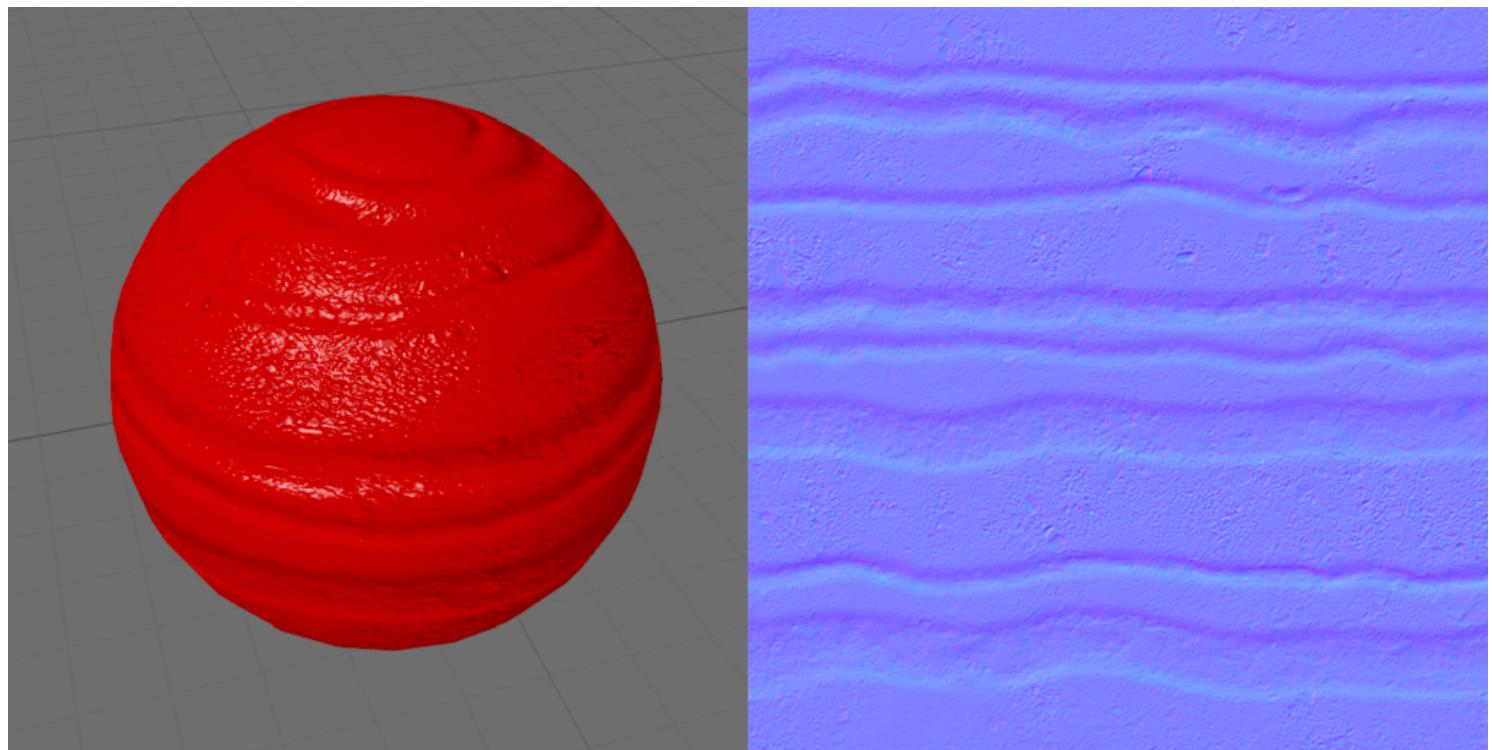


図 27.5 Normal マップ用の画像

### 27.3.5 Occlusion

Ambient Occlusion の効果を画像で行う場合使用する設定です。Xcode 上でも作成や Model I/O からも作成できますが、主に 3DCG のソフトで Ambient Occlusion のテクスチャ画像をベイクした使用します。

画像のように、光が当たらない影になる部分に部分に画像の暗部が適応されます。

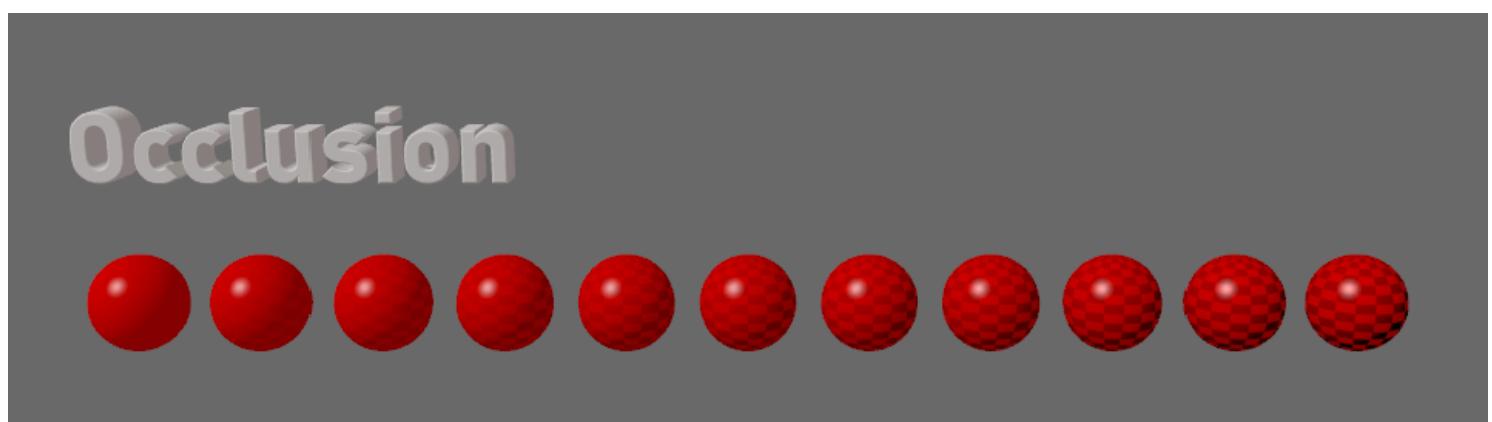


図 27.6 Occlusion

### 27.3.6 Illumination

自発光しているような見た目を表す設定で、テクスチャ画像の白い部分を元にマテリアルの彩度を上げます。



図 27.7 Illumination

### 27.3.7 Emission

こちらも自発光の値の設定ですが、ライトの陰に影響せず彩度が上がります。

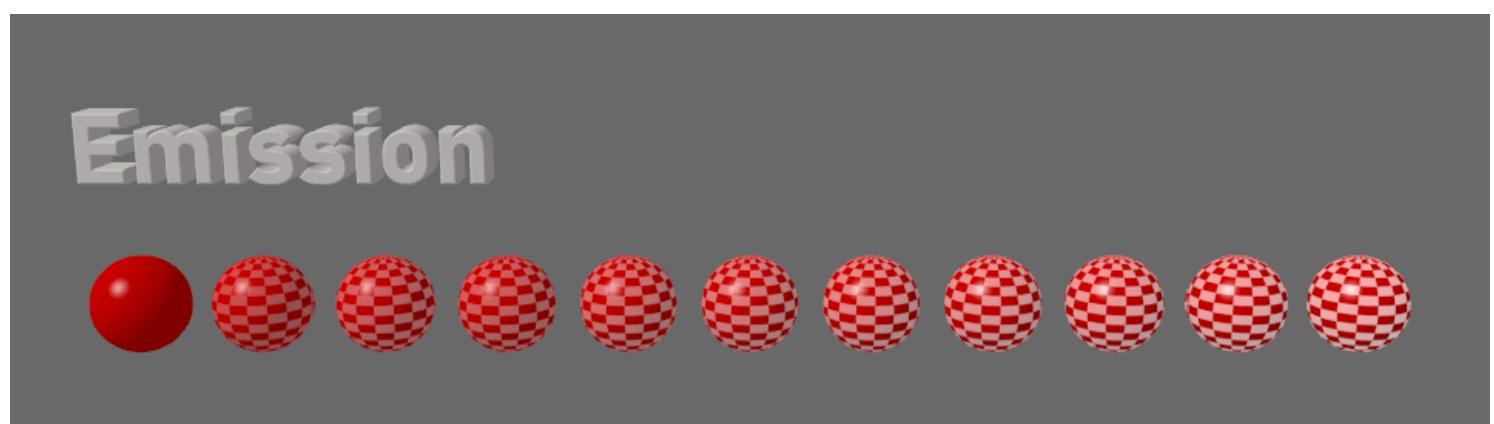


図 27.8 Emission

また、Light Probe を使用する際、こちらの設定を使用しライトの光源にすることができます。

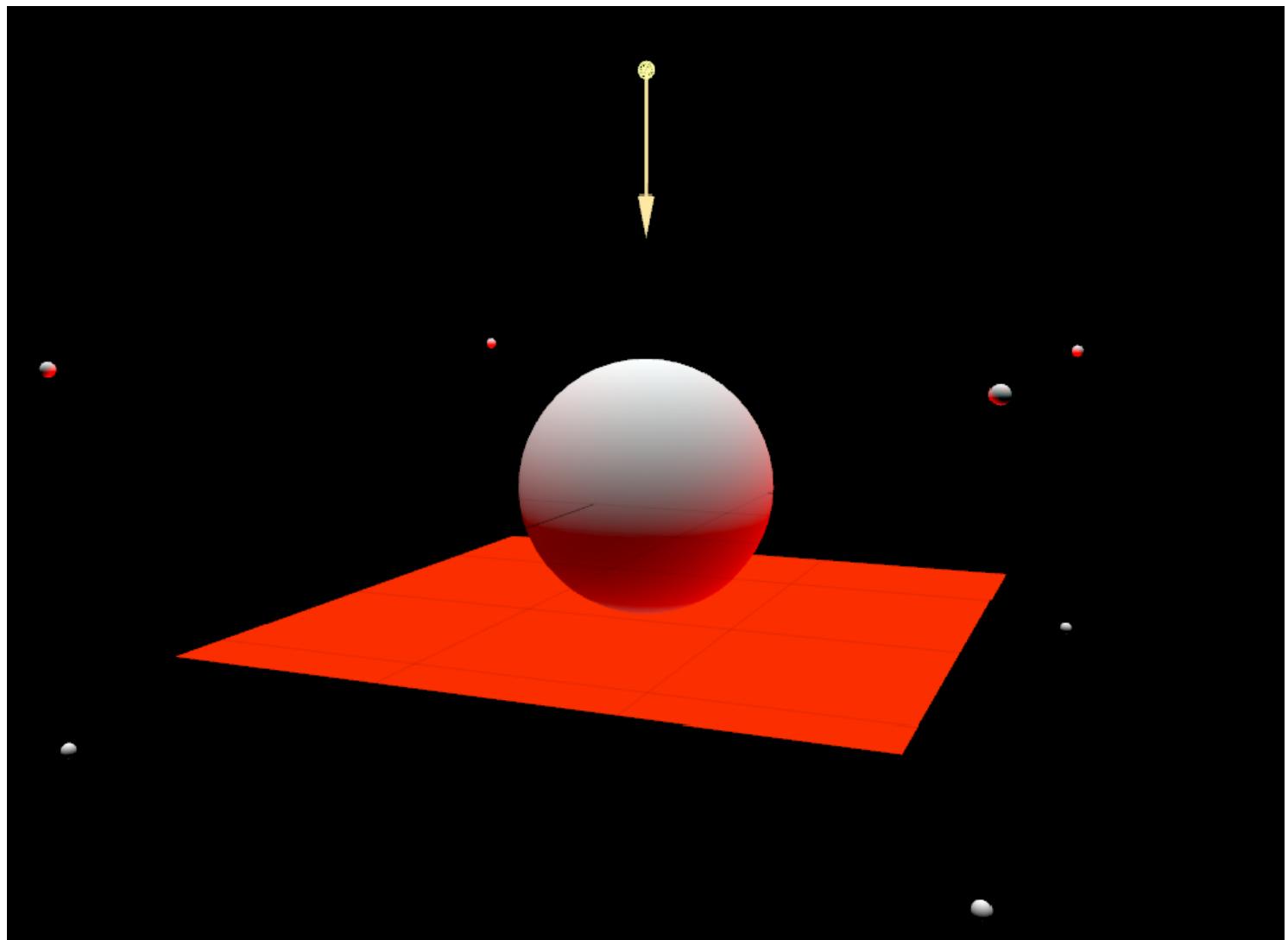


図 27.9 Emission

赤い床がライトになっているため、球体の下の部分がライトの影響を受け赤くなります。

### 27.3.8 Displacement (iOS 11 で追加)

モノクロ、またはカラーのテクスチャ画像を元にジオメトリの法線方向に頂点を移動させます。モノクロの場合は `texture components` を `red`、カラーの場合は `all` を選択します。

ジオメトリの頂点を移動させるため、きちんとしたディテールを保つ場合は、テッセレーション（要 A9 以降）かサブディビジョンサーフェイスを使用してポリコン数を増やす必要があります。

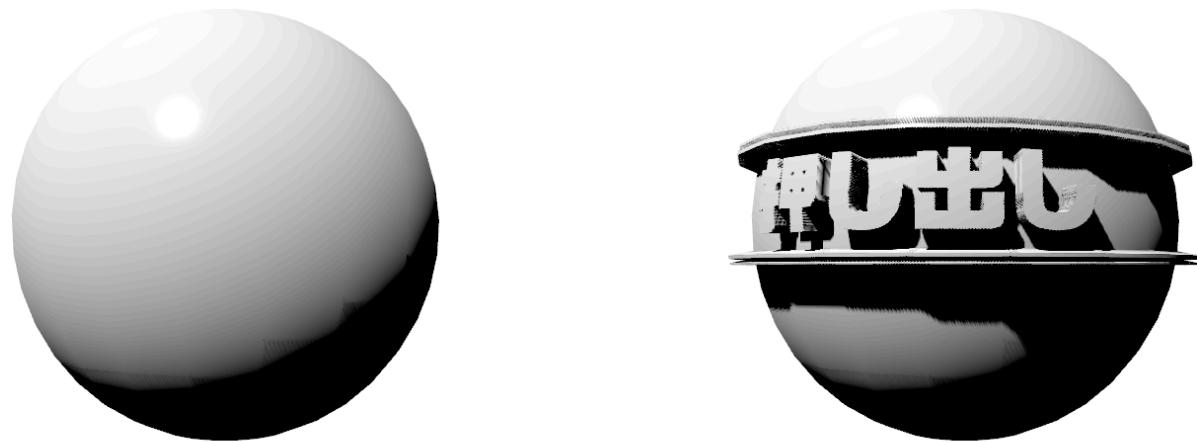


図 27.10 Displacement



図 27.11 Displacement マップのテクスチャ

現状、急激に盛り上まるものに関しては、ジオメトリを細かくしてもポリゴンのエラー出でしまいます。

# 第28章

## テクスチャ

---

すでに使用していますが、ジオメトリのマテリアルにシールを貼るように画像を設定し質感をつくります。SceneKit のテクスチャに関しては、全てマテリアルプロパティの contents で設定され使用することができます。

ここでは UIImage か NSImage の画像を受け渡す方法を紹介します。

### 28.1 SCNMaterialProperty のテクスチャ関連設定

以下のものが設定可能です。

- contentsTransform
- wrapS、wrapT
- フィルタリング設定  
(minificationFilter、magnificationFilter、mipFilter、maxAnisotropy)
- writesToDepthBuffer
- readsFromDepthBuffer
- colorBufferWriteMask
- mappingChannel

### 28.2 contentsTransform

テクスチャの大きさや傾き、表示位置などを SCNMatrix4 で行います。

## 28.3 wrapS、wrapT

ポリゴンに対して、contentsTransform でテクスチャ画像を小さくした際に余白をどう埋めていくか設定します。WrapS が横方向、WrapT が縦方向になり、以下のものが設定となります。

- Clamp
- Repeat
- Mirror

### 28.3.1 Clamp

初期値として設定されており、画像の周りの色が伸びて余白を埋めます。

### 28.3.2 Repeat

余白に画像がリピートされます。

### 28.3.3 Mirror

画像が反転されてリピートされます。偶数回数のリピート分が反転さフィルタ

## 28.4 フィルター

テクスチャを表示する際、どのようなフィルターをかけて画像の表示を設定します。

### 28.4.1 フィルタリング設定

テクスチャ表示の際にどのようにフィルターを設定するか設定値を選びます。

設定値	機能
none	設定しない
nearest	サンプリングされる座標に最も近い 1 つのテクセルを選ぶ
linear	サンプリングされる座標の近傍からサンプルのテクセルを選び線形補間する

## 28.4.2 minificationFilter

元のテクスチャ画像よりも小さく表示された場合、テクスチャの見た目を決めます。初期値は linear。

## 28.4.3 magnificationFilter

元のテクスチャ画像よりも大きく表示された場合、テクスチャの見た目を決めます。初期値は linear。

## 28.4.4 mipFilter

none ではない場合、解像度の異なる画像を作成し、遠くにあるものは低解像度のテクスチャを適応します。近くのものには高解像度し、中間のものは補完して表示するため、パフォーマンスを考慮しつつ状況に適した画像を適応できます。初期値は Nearest。

## 28.4.5 maxAnisotropy

カメラに対して極端な角度でテクスチャーが表示される場合に、テクスチャレンダリングの品質を向上させます。mipFilter が none の場合は使用することができません。

## 28.5 writesToDepthBuffer

Material の章で説明していますが、マテリアルのレンダリング時に、シーンに深度情報を生成するかどうかを指定します。false の場合、生成されないため、

## 28.6 readsFromDepthBuffer

Material の章で説明していますが、マテリアルのレンダリング時に、そのマテリアルが深度情報を使用するかどうかを指定します。false の場合、裏面が表示されたりや常に前面に表示される可能性があります。

## 28.7 colorBufferWriteMask

Material の章で説明していますが、レンダリング時にカラーバッファに書き込むかどうかを決めます。Displacement マップ使用時にグレイスケール画像か、カラー画像かを適応する場合などに使用します。

## 28.8 mappingChannel

使用する UV 情報 (Texture Coordinates) のチャンネルを選択します。ジオメトリ内で異なる UV 情報を使用してテクスチャを貼る場合、こちらに数値を渡し UV 情報を変更することで異なる面に画像を貼ることができます。

# 第29章

## キューブマップ

---

立方体 6 面の内側に指定された画像を内側に貼り付ける 360 度の背景画像をキューブマップと呼び、SCNMaterialProperty の contents へ Model I/O の機能を使用して適応します。

SceneKit ではシーンの background や lightingEnvironment、または Blinn や Phong マテリアルの Reflective プロパティで使用可能。lightingEnvironment は Physically Based のシェーダーで効果が現れます。

### 29.1 キューブマップの画像配置

ドキュメントには +X, -X, +Y, -Y, +Z, -Z で配置させると書いてありますが、ワールド座標軸的には +X, -X, +Y, -Y, -Z, +Z にしないと Z 軸に貼られる画像がおかしくなります。

また、この後に紹介する、4 つの画像はどれを適応しても同じですので、状況によって最適なものを選べばよいと思われます。

### 29.2 キューブマップ使用できる画像サイズ

名前	サイズ
Vertical strip	高さは幅の 6 倍 (height == 6 * width)
Horizontal strip	幅は高さの 6 倍の画像 (6 * height == width)
Spherical projection	幅は高さの 2 倍の画像 (2 * height == width)
Array of six images	6 枚の幅と高さが同じ画像 (height == width)

### 29.3 Vertical strip

縦並びの 1 枚画像の設定で、ドキュメントによると Vertical strip が SceneKit で使用する全てのキューブマップ処理の中で一番パフォーマンスがよいとのことです。



図 29.1 Vertical strip

## 29.4 Horizontal strip

横並びの 1 枚画像を設定します。

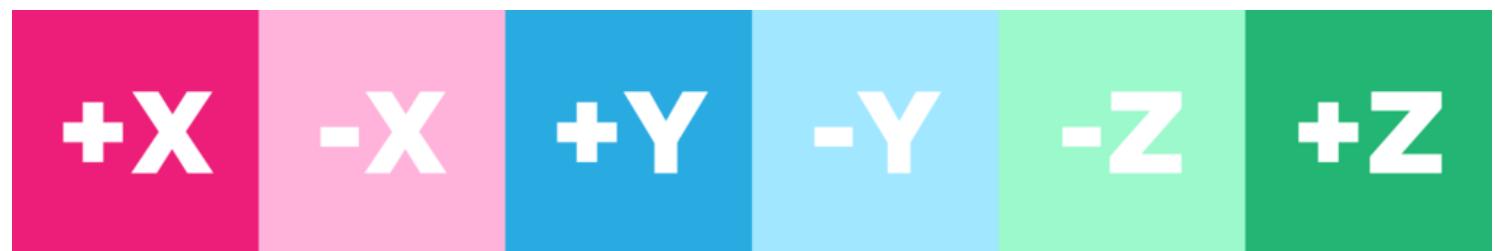


図 29.2 Horizontal strip

## 29.5 Spherical projection

1 枚のパノラマ画像を設定します。

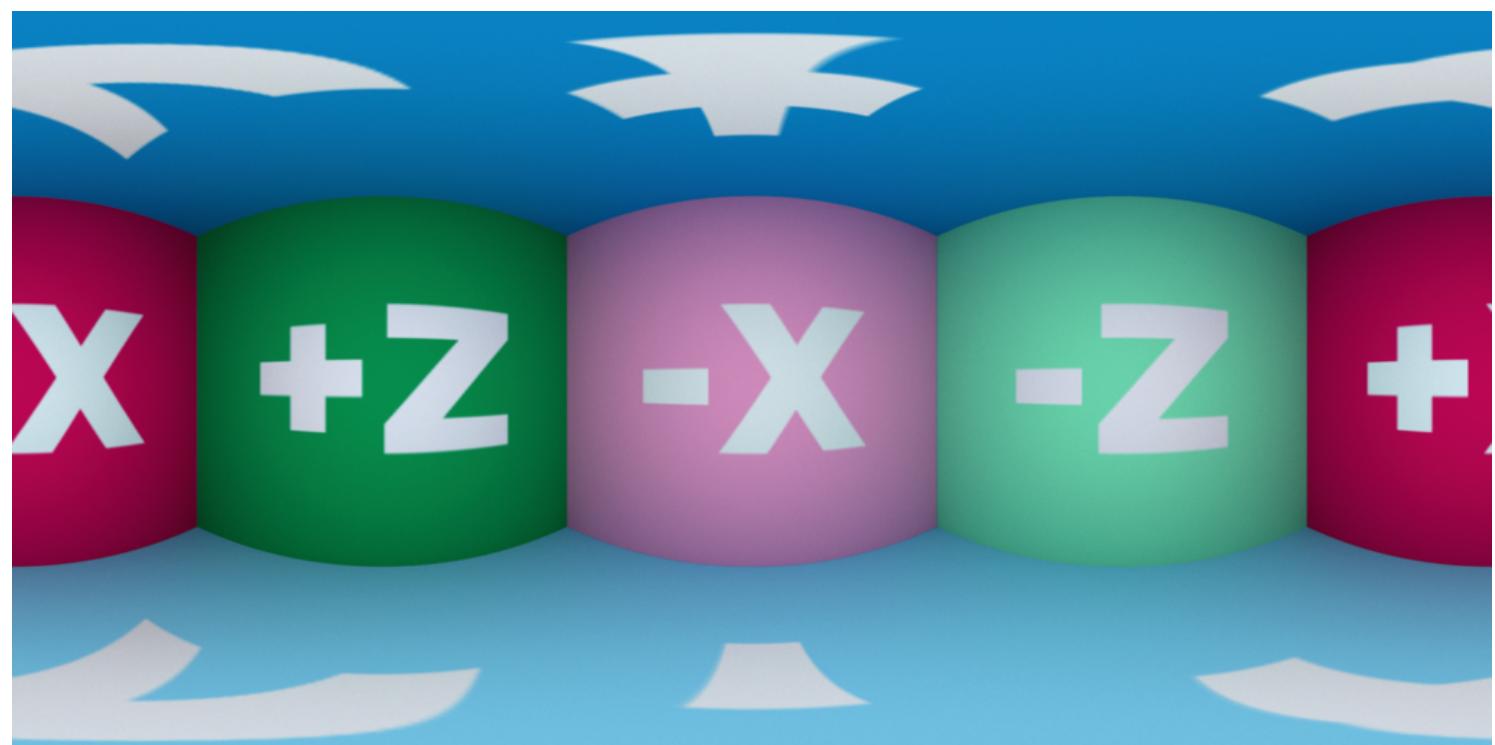


図 29.3 Spherical projection

## 29.6 Array of six images

6 枚の幅と高さが同じ画像を設定します。

リスト 29.1: 配列に 6 枚の画像を入れる

```
let scene = SCNScene()
scene.background.contents = [
    UIImage(named: "x_p.png"),
    UIImage(named: "x_m.png"),
    UIImage(named: "y_p.png"),
    UIImage(named: "y_m.png"),
    UIImage(named: "z_m.png"),
    UIImage(named: "z_p.png"),
]
```

# 第30章

## カメラ

---

これがないと何も始まらないカメラの説明。カメラ自体の設定はあまりないです。

### 30.1 カメラの設定

これまでのようにノードに `SCNCamera` を設定して `rootNode` に設定します。もし、シーン上にカメラがない場合は `SCNView` の `pointOfView` で初期設定されているカメラが動作します。

シーン内にカメラの複数設定は可能で、複数設定した場合 `pointOfView` に設定して切り替えて使用します。

### 30.2 Projection Type

設定は2つあります。

- 遠近感のある透視投影のカメラでシーンを表示する `Perspective`
- 遠近感のない平行投影のカメラでシーンを表示する `Orthographic`

`Orthographic` は設計図や図面のような遠近感がないものや、Honument Valley のような遠近感のない立体の3DCGのゲームなどに使用します。

また、`Orthographic` を指定した場合、`orthographicScale` で拡大率を決めることができます。

### 30.3 `focalLength`、`fieldOfView`、`sensorHeight`

実際のカメラはレンズからフィルム、または撮像素子までの距離で、写真や映像に映る被写体や背景の捉え方が変化します。焦点距離の `focalLength` は撮像素子までの距離、

`sensorHeight` 撮像素子の大きさで `fieldOfView` はカメラの視体積の画角となっていますが、`focalLength` と `fieldOfView` は連動しているため、どちらか片方を調整するともう片方の値が変更されます。

焦点距離と、撮像素子の大きさの単位は mm となっていて、`fieldOfView` は角度で、人間に目に人間の目に近いとされているのが、撮像素子が 35mm で焦点距離が 50mm に設定した状態です。

## 30.4 Z Clipping

カメラに映ると思定される部分を全て GPU で描画するとマシンパワーを使いすぎるので、大概の 3DCG では手前と奥をどこまで描画するか `zNear`、`zFar` で決めます。

`zNear` はその設定値より前、`zFar` はその値より後が描画されなくなり、特に `zFar` は部屋など、遠方を表示しない場合に値を小さくすると処理が軽くなります。

`automaticallyAdjustsZRange` を使用すると自動的に `zNear`、`zFar` を設定します。`zNear`、`zFar` のいずれかの設定値を変更するとこの設定は無効になります。

## 30.5 被写界深度

ピントのあってないものをぼかすエフェクトを施しますが、残念ながらここで紹介するプロパティは `wantsDepthOfField` 以外 iOS 11 から廃止予定なっており、個人的な予想では、新たなプロパティが加わるか、今回変更された焦点距離などから類推されるものだと思われます。

## 30.6 モーションブラー

モーションブラーは、カメラがものを捉える際、ローリングシャッターの問題で起きる現象で一定速度以上になった場合映る残像のこと。カメラの撮影、録画での手ブレとかがわかりやすい例かと思われます。

設定値 `motionBlurIntensity` の初期値は 0 で、0 以上にすると効果が現れます。

## 30.7 iOS 11 追加されたが設定がわからないもの

視野の設定 `projectionDirection` で `horizontal` か `vertical` を設定することができますが、何が変わっているかわかりませんでした。

# 第31章

## ライト

---

シーン上には必ずライトとなるものが1つ必要となります。SCNNodeでlightプロパティにSCNLightの設定を行うか、シーンの章のLight Environment (Image Based Lighting / IBL)を設定することで、ジオメトリのマテリアルに光を照らしカメラに物体を映すことができます。

### 31.1 SCNLight の種類

ライトの種類は以下のものとなります。

- オムニライト（豆電球のように指定位置から全方向に照らす）
- ディレクショナルライト（位置に関係なく指定した方向に照らす）
- スポットライト（指定位置から全方向に照らす）
- アンビエントライト（位置に関係なくシーン全体を照らす）
- IESライト（照明をシミュレーションしたIESファイルを使用して照らす）
- Light Probe（ジオメトリのマテリアルEmissionをライトに使用する）

### 31.2 SCNLight の制限

ドキュメントではSCNLightはノードで使用できる数が8つとなっています。

通常、ライトを8つ使用することはあまりませんが、試したところ、アンビエントライト、光の変更を行わないモードであるスタティックにしたライト、ライトプローブを除くライトが8個以上あると9個目からは設定した順番の古いライトからライトの適応が無視されます。

### 31.3 SCNLight の注意点

3DCG はシーン内のライトの数と種類でレンダリングパフォーマンスが変化し、SceneKit で効率的なレンダリングを行う場合、シーン内のライティング設定を見直す必要があるります。

#### 調整例

- 光の当たり方が固定の場合は、あらかじめ 3DCG のオーサリングツールでライトをライトマップテクスチャにベイクし、SCNMaterial にそのテクスチャ適応して multiply で色等を調整する
- 必要のないライトやジオメトリで影の描画を行わない
- ゲームキャラクタなど正確な影の形が必要ない場合、影を輪郭がぼやけた丸などの画像にしてしまう
- スポット、オムニ、ディレクショナルライトで attenuationEndDistance プロパティを設定し有効範囲を制限する
- SCNLight の categoryBitmask 使用してライトを必要ないジオメトリにはライトを適応しない

### 31.4 SCNLight の共通設定

いくつかの共通設定があります。Light Probe のみ設定できない項目もあります。

#### 31.4.1 Type

ライトの種類を SCNLight.LightType を使用し設定します。

type の設定後、IES ライトのファイルを読み込む関数 iesProfileURL を使用すると、何が設定されていても IES ライトに変更され流るので注意してください。

表 31.1 Type

プロパティ	名前
ambient	アンビエントライト
omni	オムニライト
directional	ディレクショナルライト
spot	スポットライト
IES	IES ライト
probe	Light Probe

## 31.4.2 Mode

初期値は Dynamic でシーン内のジオメトリが動いても光やマテリアルの情報が再計算されます。

もう一つの値の Static ですが動的に変更されることはありません。画面を更新しない場合やライトの光をテクスチャにベイクする場合に使用しますが Swift Playgrounds 上ではほぼ使われません。

## 31.4.3 Color

ライトの色を設定します。

この色とともにジオメトリのマテリアルを使用して最終的な色を設定します。

## 31.4.4 Intensity

光の強さとなっており、初期値は 1000。マテリアルのライティングモデルによっては、この値を上げると光沢の範囲が増えたり、光沢の輪郭がシャープになります。

## 31.4.5 temperature

あまり使用することはないと思われますが、色温度の単位ケルビンでライトの色設定を行うことができます。6500 K で白色となっており、低くすると赤みがかり、高くすると青くなります。初期値は 6500 になっており、設定できる値は 0 ~ 40000 です。

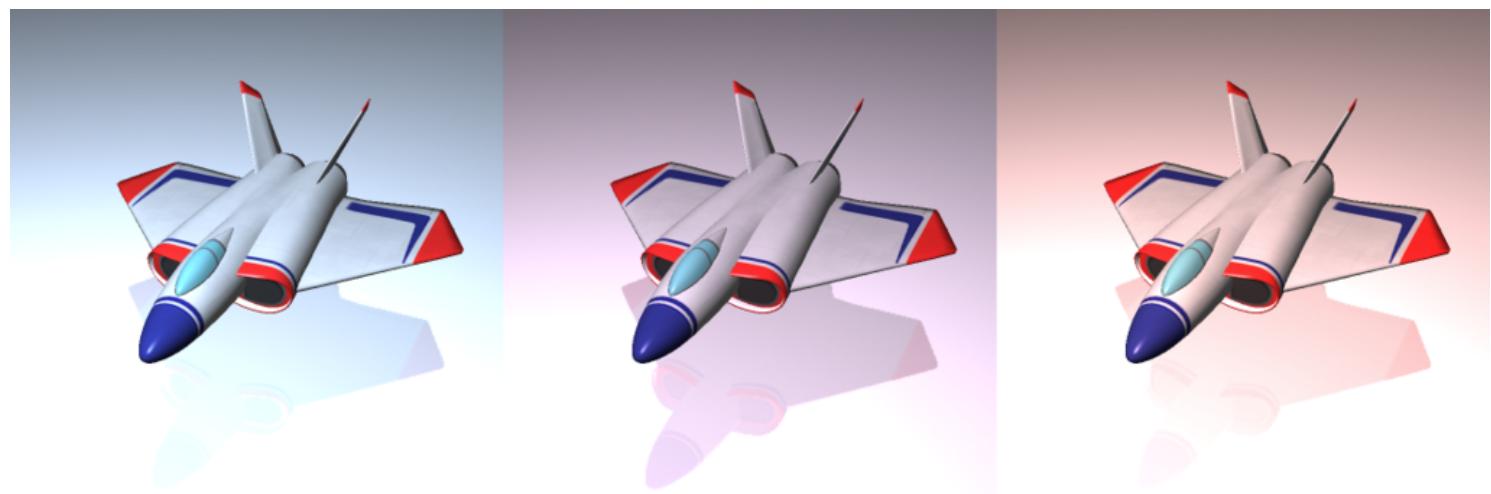


temperature 6500

temperature 3000

temperature 10000

図 31.1 temperature の適応



temperature 10000  
intensity 1500  
color 設定なし

temperature 10000  
intensity 1500  
color 設定あり

temperature 設定なし  
intensity 1500  
color 設定あり

図 31.2 temperature と intensity との併用

### 31.4.6 その他

初期化の際、Model I/O で設定したライトを適応することができます。

## 31.5 Omni Light と Attenuation 設定

Omni Light は設置位置によって光の影響を変化させられるが、ジオメトリの影を落とすことはできません。強い光など、影は必要ないが光の影響を与えたい場合などで使用されると思われます。

### 31.5.1 Attenuation

直訳すると減衰となり、光がどう減衰するかを設定します。

#### attenuationStartDistance

減衰の開始の距離となり、初期値は 0.0 でライトの設置場所から行われます。

#### attenuationEndDistance

減衰の終了距離。これ以降は光の影響を受けません。

デフォルト値が 0.0 で適応範囲は無限遠となっており、かなりなだらかな減衰となります。

#### attenuationFalloffExponent

減衰の状態を表し、初期値が 2.0 です。

値が 0 の場合は完全に減衰は行われません。

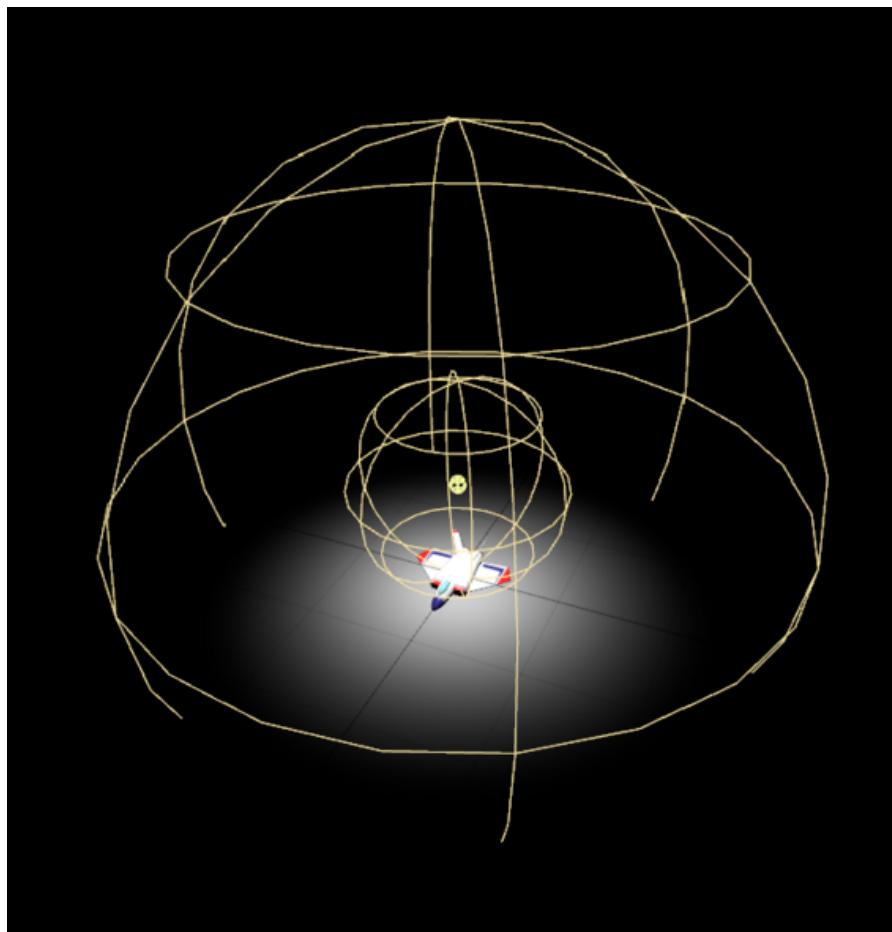


図 31.3 Attenuation 設定例

## 31.6 Ambient Light

設置位置に関係なく光の影響を与え、ジオメトリの暗部である陰の色を薄めて画面全体の色を調整することができます。

## 31.7 Spot Light とキャストシャドウ（シャドウマップ）

舞台照明で使用されているスポットライトを模したライトです。設置位置から円錐状に光の影響を与え、遮るジオメトリがあった場合に影を与えるキャストシャドウを設定することができます。

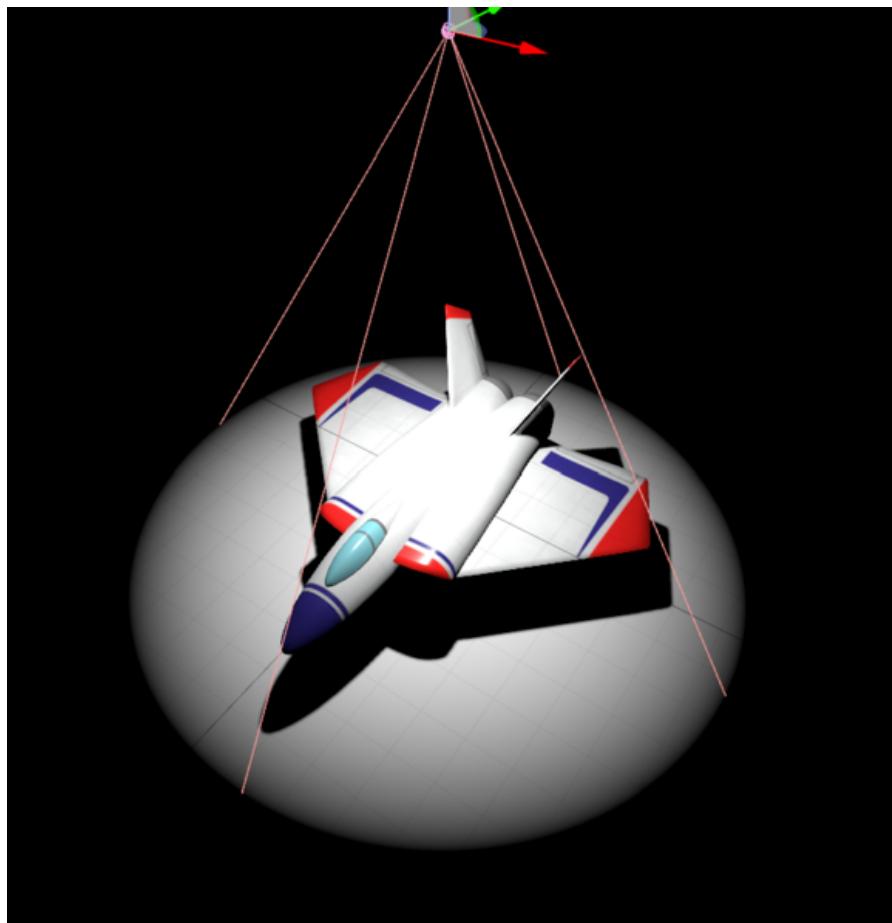


図 31.4 Spot Light に照らされキャストシャドウが落ちている

### 31.7.1 Attenuation 以外の個別設定

#### Outer angle

円錐の円の大きさを設定します。初期値は 45 度。

#### Inner angle

円錐の内側の円を決めることで光の減退を設定することができます。初期値は値は 0 度で影響がでません。

### 31.7.2 Casts Shadows

#### Color

影に色を設定し、半透明も使用できます。初期値は黒で、アルファは 1 の不透明になっています。

後ほど説明する Mode で「Deferred」を選択しないと設定した色は反映されずに、影は黒になります。

#### Sample radius

影の輪郭のシャープさを設定し値を小さくするとシャープになり、大きくすると輪郭がぼやけます。初期値は 3.0 になります。

#### Mode

影の処理するモードを設定します。

表 31.2 Mode

値	説明
Forward	デフォルト値。そのままシャドウマップを表示する color で設定しても色は適応されないが透明度は適応される
Deferred	最終画像をレンダリングした後に color で設定している影の色を適応する
Modulated	任意の画像の影 (gobo) を使用し影を描画するモード 画像設定はコードから行う

#### その他

シャドウマップの画像に対する設定がありますが、iOS 11 で追加された automaticallyAdjustsShadowProjection を true (初期値) に設定すると自動で計算してくれます。

また、iOS 11 で追加されたカスケードシャドウマップの設定に関してはページ数と説明のしづらさの問題があるため割愛させてもらいます。

## 31.8 Directional Light

太陽光のように一定の方向に光を放ち、設置位置に関わらず一定で光の影響を与えるライトです。スポットライト同様に遮るジオメトリがあった場合に影を与えるキャストシャドウを設定することができます。

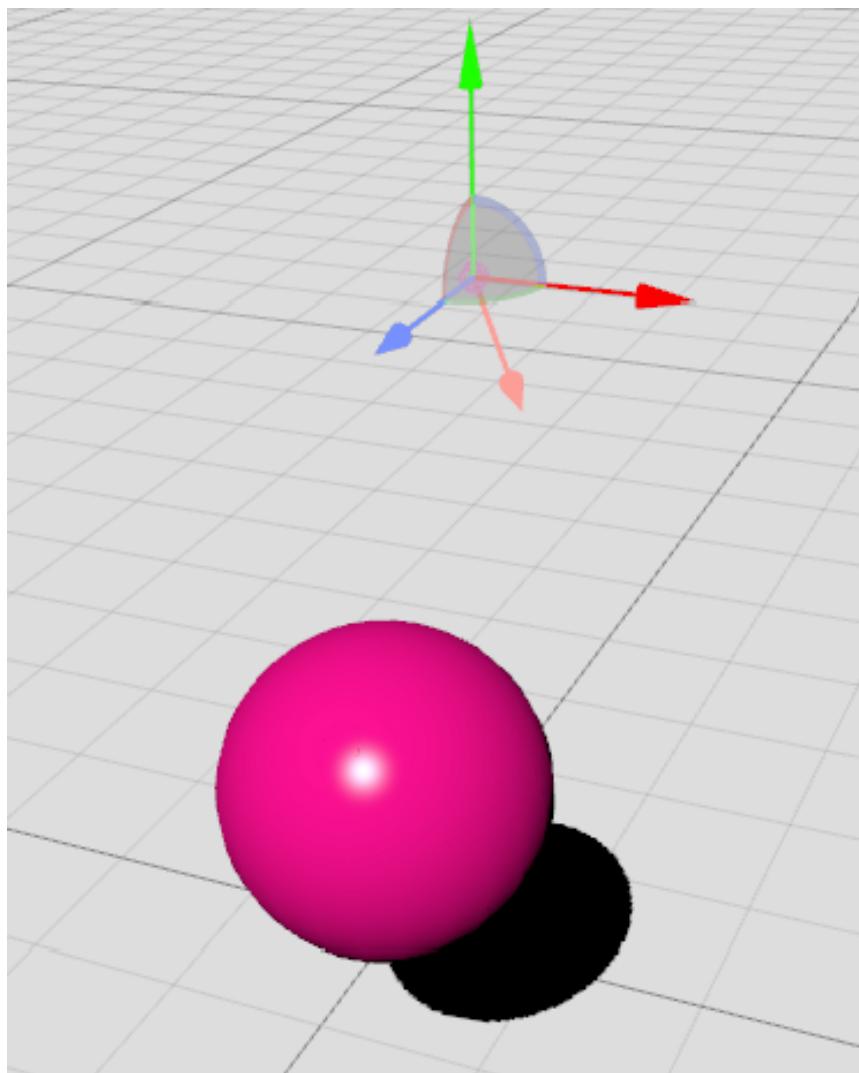


図 31.5 Directional Light

## 31.9 IES Light

テキストデータのファイルを使用し照明のシミュレーションを行います。Directional Lightとほぼ同じで、IES ファイルを読み取る命令があります。

IES ファイルですが照明メーカー、レンダラーを開発しているサイト、3DCG DCC ツールを販売しているサイトなどで配布しています。サンプルでは Resources > IES にファイルがあります。

リスト 31.1: IES を適応したライト

```
let light = SCNLight()
light.type = .IES
light.intensity = 1000

let lightNode = SCNNNode()
lightNode.light = light
lightNode.position = SCNVector3(x: 0, y: 5, z: 0)
lightNode.eulerAngles = SCNVector3(x: CGFloat(-Float.pi * 0.5), y: 0, z: 0)
scene.rootNode.addChildNode(lightNode)

let iesPath:String = Bundle.main.path(forResource: "iesprofile", ofType: "ies")!
let fileURL:URL = URL(fileURLWithPath: iesPath)
light.iesProfileURL = fileURL
```

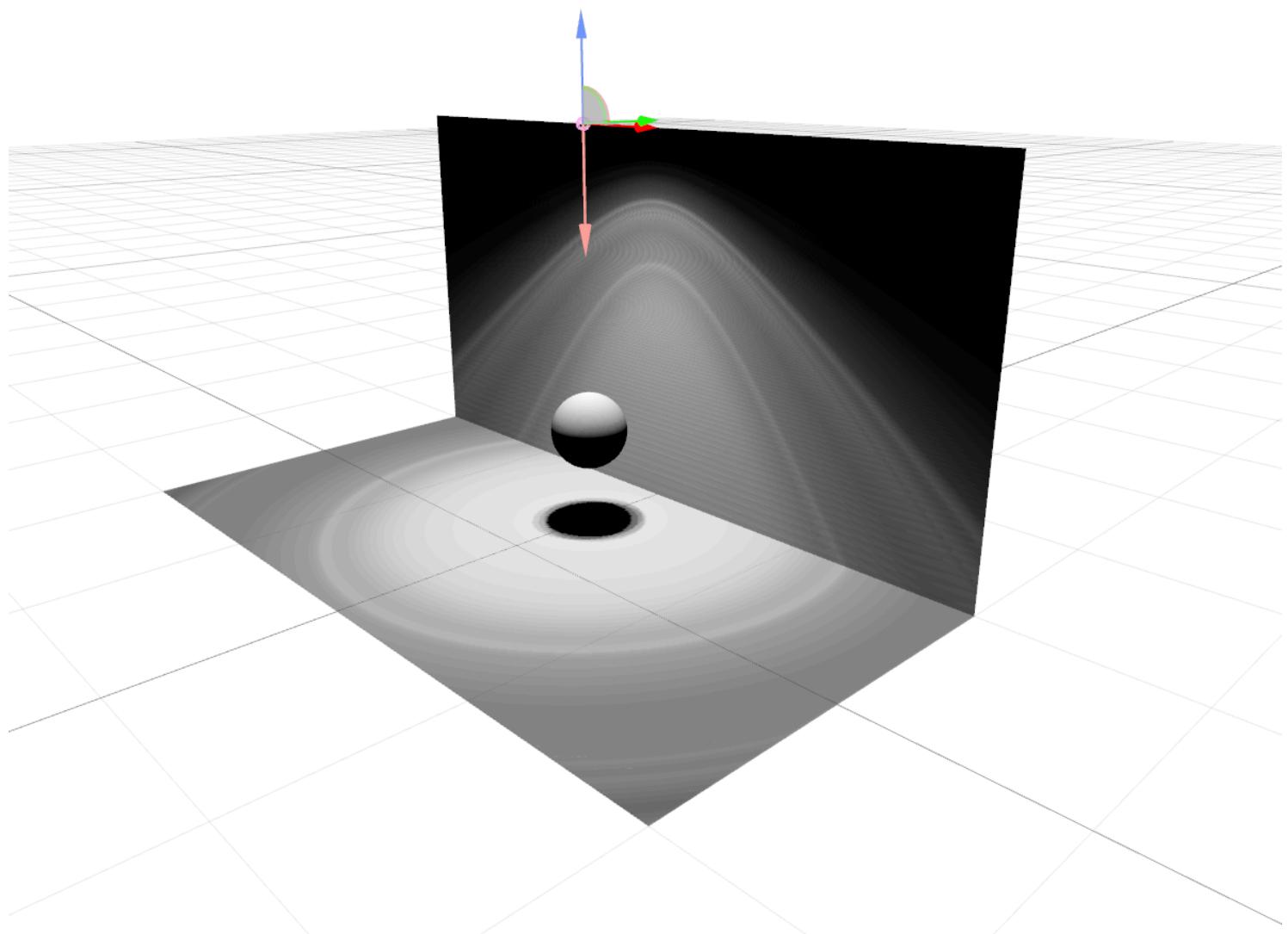


図 31.6 IES を適応した画像

動作的ライトからのシャドウマップ使用し表現しているため、アーティファクトが出る（データのエラーやノイズの出現する）場合があります。

その場合は、Casts Shadow の チェックをオンにして Shadow のパラメータ群で調整する必要があります。

### 31.9.1 注意点

ライトの種類を選択後 IES ファイルの読み込みが呼ばれると Omni、Spot など他のライトを選択していても IES に変更されます。

また、オフラインレンダリングで使用する用途で作成されているため、モバイルの GPU では記述されている IES が意図通りに表示されない可能性があります。

## 31.10 Light Probe

Light Probe は事前計算する必要があるため、本書で詳しい動作方法は説明できませんが、以下が Light Probe の概要となります。

### 31.10.1 概要

シーンで動かない物体にはライトマップでテクスチャに焼き付けると綺麗な見た目となります、キャラクターなど動くものがあった場合、光は反映されません。

Light Probe はシーンの色の変化とすべての方向から受け取る照明の強度を球体で設定し、有効範囲にある Physically Based のライティングモデルの Emission の値を光の情報として使用します。

そのため適応させたいキャラクターなどのジオメトリが動作する周辺にいくつか設置する必要があります。

### 31.10.2 Light Probe の制限

- ライトとなるジオメトリとそれを適応させるジオメトリは Physically Based のライティングモデルである必要
- 以前の記事で書いたが Light Probe は自体がライトになるわけではなく、ジオメトリの Emission の値がライト
- Emission の光を反映させたいジオメトリは Movability を true にする
- キャストシャドウは落とせないので他のライトと併用する

# 第32章

## 物理アニメーションの概要

---

この章から物理アニメーションについていくつか解説をしていきます。

物理アニメーションとパーティクルに関しては設定する値が多く、相互的に多くのパラメーターが関係しているため、申し訳ないのですが内容が複雑になっています。

また、用語が増えるとわかりづらくなる可能性もあるため、本書では物理シミュレーションも物理アニメーションとして記載しています。

### 32.1 物理アニメーションで設定するもの

パーティクルシステムを除くと SceneKit の物理アニメーションで使用するシミュレーションは大きく分けると以下のものになります。

- PhysicsWorld
- PhysicsBody
- PhysicsField

PhysicsWorld は物理アニメーション全体についての設定、

PhysicsBody は物理アニメーションをさせる対象物、

PhysicsField は風などの外部からの力を指します。

#### 32.1.1 その他

物理アニメーションにはこれ以外にも PhysicsBody の当たり判定を決める PhysicsShape、PhysicsWorld 内で物が当たった時に作用する PhysicsContact とその Delegate、車の動きをシミュレーションする PhysicsVehicle、コンストレイントに属するヒンジやボールジョイントなどをシミュレーションする Joint があります。

## 32.2 物理アニメーションの種類

SCNNode であれば全て物理アニメーションが可能ですが、演算処理の観点から、物理アニメーションを行う際に状況に合わせた種類を以下の3つから選択する必要があります。

- Static
- Dynamic
- Kinematic

### 32.2.1 Static

設置した部分に固定され、他の PhysicsBody の力や衝突の影響を受けません。例えば、物理アニメーションの際に固定されて欲しいと思われる床や部屋などで使用します。

### 32.2.2 Dynamic

すべての力や衝突の影響を受けます。主に使用するものがこちらで、物理アニメーションを有効し、固定させておきたいもの以外はすべてこれを使うことになります。

### 32.2.3 Kinematic

Static 同様に画面に固定されますが、このノードが動いた時に物理アニメーションが動作する。

例えば、Static で床をつくった場合、物理アニメーションの判定は最初に設置した状態で当たり判定を固定します。球を Dynamic にし床へ落とした場合、Static の床を X 軸 45 度に傾けても球はその場から動くことはありません。床を Kinematic で設定した場合は固定せながら、動かしても物理アニメーションが有効になるため、床を 45 度動かすと見た目通り球は落下します。

## 32.3 処理負荷

負荷的には Dynamic > Kinematic > Static となっていて、Static にできるものはできるだけ Static で設定した方がよいです。

## 32.4 物理アニメーションを行う

以下のコードでは、static で設定された SCNFloor の床をつくり、タップするたびに球が落ちるようになっており、 $10 \pm 4$  秒で消滅します。負荷を軽減するため、viewDidLoad で球を作成し、タップ時に複製してシーンに配置します。

リスト 32.1: 簡単な物理アニメーションサンプル

```
import PlaygroundSupport
import SceneKit

class GameViewController: UIViewController {

    var ballNode:SCNNode!
    var scnView:SCNView!

    override func viewDidLoad() {
        super.viewDidLoad()

        // シーン
        let scene = SCNScene()

        // 床
        let floor = SCNPlane()
        let floorNode = SCNNNode(geometry: floor)
        floorNode.position = SCNVector3(0, -4, 0)
        floorNode.physicsBody = SCNPhysicsBody(type: .static, shape: nil)
        scene.rootNode.addChildNode(floorNode)

        // オムニ ライト
        let lightNode = SCNNNode()
        lightNode.light = SCNLight()
        lightNode.light!.type = .omni
        lightNode.position = SCNVector3(x: 0, y: 10, z: 10)
        scene.rootNode.addChildNode(lightNode)
    }
}
```

```
// アンビエント ライト
let ambientLightNode = SCNNode()
ambientLightNode.light = SCNLight()
ambientLightNode.light!.type = .ambient
ambientLightNode.light!.color = UIColor.darkGray
scene.rootNode.addChildNode(ambientLightNode)

// カメラ
let cameraNode = SCNNode()
cameraNode.camera = SCNCamera()
cameraNode.position = SCNVector3(x: 0, y: 0, z: 10)
scene.rootNode.addChildNode(cameraNode)

// View 設定
scnView = SCNView()
self.view.addSubview(scnView)

// View の Autolayout
scnView.translatesAutoresizingMaskIntoConstraints = false
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "V:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
    "H:[scnView]|", options: NSLayoutConstraintOptions(rawValue: 0),
    metrics: nil, views: ["scnView": scnView]))

scnView.scene = scene
scnView.allowsCameraControl = true
scnView.showsStatistics = true
scnView.backgroundColor = UIColor.black

// 球のジオメトリ
let ball = SCNSphere(radius: 0.5)
ballNode = SCNNode(geometry: ball)
ballNode.position.y = 4

ballNode.runAction(SCNAction.sequence([
    SCNAction.wait(duration: 10, withRange:4),
    SCNAction.fadeOut(duration: 1),
    SCNAction.removeFromParentNode()
])))

// 球の物理アニメーション設定
let physicsBall = SCNPhysicsShape(node: ballNode, options: nil)
```

```
ballNode.physicsBody = SCNPhysicsBody(type: .dynamic, shape: physicsBall)

// タップジェスチャー
let tapGesture = UITapGestureRecognizer(target: self,
action: #selector(handleTap(_:)))
scnView.addGestureRecognizer(tapGesture)
}

objc
func handleTap(_ gestureRecognize: UIGestureRecognizer) {
    // シーンに球を複製する
    scnView.scene?.rootNode.addChildNode(ballNode.clone())
}

override var shouldAutorotate: Bool {
    return true
}

override var prefersStatusBarHidden: Bool {
    return true
}

override var supportedInterfaceOrientations: UIInterfaceOrientationMask {
    if UIDevice.current.userInterfaceIdiom == .phone {
        return .allButUpsideDown
    } else {
        return .all
    }
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
}

}

PlaygroundPage.current.liveView = GameController()
```

# 第33章

## 物理アニメーションの共通設定

---

物理アニメーション時に PhysicsBody であるジオメトリがどのような特性を持っているか設定したり、PhysicsWorld による物理アニメーションの全体設定をします。

### 33.1 質量、抵抗、電荷

ノードの PhysicsBody で設定できる 質量、抵抗、電荷 などの値です。

- Mass (Dynamic, Kinematic のみ)
- Friction
- Restitution
- Rolling friction
- Damping
- Angular damping
- Charge
- Gravity (isAffectedByGravity)
- Resting (allowsResting)

#### 33.1.1 Mass

質量を表します。初期値は 1 で、1kg という設定担っており、0 で静止します。

例えば、他の Dynamic の PhysicsBody が同じ速度でぶつかった場合、質量が高い方が跳ね返す力が強くなります。

#### 33.1.2 Friction

設置面の荒さによって抵抗を設定する摩擦の値で初期値は 0.5 です。

対象の PhysicsBody の Friction も 0 の場合は摩擦抵抗がなくなり滑り続け、互いが 1.0 の場合は全く滑らなくなります。

### 33.1.3 Restitution

他の PhysicsBody と衝突する際に運動エネルギーをどれだけ失うか、またはどれだけ運動エネルギーを得るかを設定します。初期値は 0.5。

1 以上にすると衝突の際に力が発生するため、物体が弾みます。そのため、SCNFloor を Static な PhysicsBody で設定し、Dynamic の SCNSphere を上から落とした際、初期状態ではそのまま床に張り付きますが、Restitution を 1 以上にすると SCNSphere が弾みます。

### 33.1.4 Rolling friction

回転運動に対する摩擦による抵抗力を表し、初期値は 0。

設置面の床と回転して落ちる球の Rolling friction の値を大きくすると、設置時に回転せずに止まるようになります。

### 33.1.5 Damping

流体摩擦、または空気の摩擦抵抗の影響をシミュレートし、初期値は 0.1。

0.5 になると空間での抵抗を受けゆっくり落ち、1.0 になると静止します。

### 33.1.6 Angular damping

Damping は移動に対しての抵抗でしたが、こちらは回転に対して抵抗を与えます。初期値は 0.1。

0.5 になると空間での抵抗をゆっくり受け回転し、1.0 になると全く回転はしなくなります。

### 33.1.7 Charge

PhysicsBody が持つ電荷を設定します。初期値は 0.0 で電場や磁場の影響を受けません。

SCNPhysicsField の Electric Field、または Magnetic Field などの磁力に関する外圧を使用する際に影響を受け、負の値はマイナス、正の値はプラスとなります。

### 33.1.8 Gravity (isAffectedByGravity)

isAffectedByGravity を true にすると PhysicsWorld で設定されている重力の影響を受けます。そのため Dynamic の PhysicsBody で true にすると、重力の影響を受けないため静止してしまいます。初期値は true。

### 33.1.9 Resting (allowsResting)

isResting というプロパティが存在しており、力の影響を受けていない場合、SceneKit の物理シミュレーションによって、自動的に true に設定され停止状態になります。allowsResting のチェックをオンにすることで、自動的に isResting を false に戻してシミュレーションさせるか否かを決めます。初期値は true。

## 33.2 質量、抵抗、電荷の設定例

リスト 33.4: 設定例

```
let ball = SCNSphere(radius: 1)
let ballNode = SCNNNode(geometry: ball)
ballNode.position.y = 10
ballNode.physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)

ballNode.physicsBody?.mass = 1.0
ballNode.physicsBody?.friction = 0.5
ballNode.physicsBody?.restitution = 0.5
ballNode.physicsBody?.rollingFriction = 0.0
ballNode.physicsBody?.damping = 0.1
ballNode.physicsBody?.angularDamping = 0.1
ballNode.physicsBody?.charge = 0.0
ballNode.physicsBody?.isAffectedByGravity = true
ballNode.physicsBody?.allowsResting = true

scene.rootNode.addChildNode(ballNode)
```

## 33.3 速さやその力

ノードの PhysicsBody で設定できる Velocity についての解説です。Static は力が加えられる事はないのでこの設定はできません。

### 33.3.1 Linear velocity

物理アニメーション時に、現在の移動に加え指定した方向へ力を与えます。初期値は x: 0.0 y: 0.0 z: 0.0。x を 1 にすると、プラス方向に メートル/秒 で移動の力が働きます。

値を読み取る際は SCNSceneRendererDelegate プロトコルのアニメーションループの状況に異なるため注意が必要です。

### 33.3.2 Angular velocity

物理アニメーション時に、現在の回転に加え指定した方向へ力を与えます。初期値は x: 0.0 y: 0.0 z: 0.0。x を 1 にすると、プラス方向に メートル/秒 で回転の力が働きます。

Linear velocity 同様に、値を読み取る際は SCNSceneRendererDelegate プロトコルのアニメーションループの状況に異なるため注意が必要です。

### 33.3.3 Linear factor

物理アニメーションの移動方向に制限を与えます。初期値は x: 1.0 y: 1.0 z: 1.0 x: 1 y: 1 z: 0 に設定すると Z 軸方向の移動が行われなくなり、2 次元平面を移動しているようになります。

### 33.3.4 Angular factor

物理アニメーションの回転方向に制限を与えます。初期値は x: 1.0 y: 1.0 z: 1.0 x: 0 y: 1 z: 0 に設定すると Y 軸方向のみ回転します。

## 33.4 速さやその力の設定例

リスト 33.4: 設定例

```
let ball = SCNSphere(radius: 1)
let ballNode = SCNNNode(geometry: ball)
ballNode.position.y = 10
ballNode.physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)

ballNode.physicsBody?.velocity = SCNVector3(0,0,0)
ballNode.physicsBody?.angularVelocity = SCNVector4(0, 0, 0, 1)
ballNode.physicsBody?.velocityFactor = SCNVector3(1, 1, 1)
ballNode.physicsBody?.angularVelocityFactor = SCNVector3(1, 1, 1)

scene.rootNode.addChildNode(ballNode)
```

## 33.5 慣性モーメント

回転する物体がその回転運動を持続しようとする慣性の大きさを表わす量となり、SceneKitでは形状と質量に合わせて自動設定されます。

`usesDefaultMomentOfInertia` を `false` にして、`momentOfInertia` を設定すると独自の慣性モーメントを付加できます。

## 33.6 慣性モーメントの設定例

SceneKit の自動設定を解除し、X 軸回転方向に制限をかけます。

リスト 33.3: X 軸回転方向に制限をかける

```
let ball = SCNSphere(radius: 1)
let ballNode = SCNNNode(geometry: ball)
ballNode.position.y = 10
ballNode.physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)

ballNode.physicsBody?.usesDefaultMomentOfInertia = false
ballNode.physicsBody?.momentOfInertia = SCNVector3(0, 1, 1)

scene.rootNode.addChildNode(ballNode)
```

## 33.7 力、トルク、衝動（力積、インパルス）

力、トルク、衝動の3つに対して実行する命令とキャンセルする命令があります。ユーザー操作で PhysicsBody を動かす場合はこちらを使用します。

- `applyForce(SCNVector3, asImpulse: Bool)`
- `applyForce(SCNVector3, at: SCNVector3, asImpulse: Bool)`
- `applyTorque(SCNVector4, asImpulse: Bool)`

### 33.7.1 動作

2つは移動の力となり、`applyTorque` は回転であるトルクの力が現在与えられている力に加わ割ります。1つ目の `applyForce` はジオメトリの重心から力を与え、2つ目の `applyForce` の `at` はノードのローカル座標の場所から力を与えます。

`asImpulse` を `true` にすると運動量の瞬間的な変化が適用され、`false` にすると最初のアニメーションループ（動作1フレーム目）にのみ適応されている模様です。

### 33.7.2 単位（通常）

- Force の単位はニュートン
- Torque の単位はニュートンメートル

### 33.7.3 単位（`asImpulse` が `true` の場合）

- Force の単位はニュートン秒
- Torque の単位はニュートンメートル秒

## 33.8 力、トルク、衝動（力積、インパルス）の設定例

リスト33.4: 設定例

```
node.physicsBody?.applyForce(SCNVector3(1,0,0), asImpulse: true)
node.physicsBody?.applyForce(SCNVector3(1,0,0), at: SCNVector3(0,0,0),
asImpulse: true)
node.physicsBody?.applyTorque(SCNVector4(0,0,-1,1), asImpulse: true)

// Force や Torque のキャンセル
node.physicsBody?.clearAllForces()
```

## 33.9 物理アニメーションの全体設定

シーンには SCNPhysicsWorld というものがあり、物理アニメーションを全体の設定決めるものと、コンストレインとを用いたジョイントなどが設定できます。

全体の設定で主に使用するものは以下のものですが、基本的には初期値のままで問題ありません。

- gravity
- speed
- timeStep
- updateCollisionPairs()

### 33.9.1 gravity

シーン全体の重力を SCNVector3 で設定します。Y 軸下向き方向で設定されていますが、宇宙空間や演出で重力を変更したい場合に使用できます。

### 33.9.2 speed

物理アニメーションのアニメーションスピードを設定します。初期値が 1 なので、2 になると倍のスピードでアニメーションし、0 にすると止まります。

### 33.9.3 timeStep

物理アニメーションのフレームレートを変更することができ、初期値は 1/60 秒（60 フレーム）。変更する場合は SCNView でフレームレートを設定している preferredFramesPerSecond に一致するようにする必要があります。

### 33.9.4 updateCollisionPairs()

PhysicsBody 間の衝突処理を再評価させる関数。SCNPhysicsContactDelegate のメソッドの適応範囲外にあるものを操作し物理アニメーションをする場合、設定した delegate での衝突判定を通らない可能性があるため、この関数を呼び出し即時適応させます。

# 第34章

## 物理アニメーションのカテゴリービットマスク

---

PhysicsBody には SCNNode のビットマスクとは別に、当たり判定判別するため複数のビットマスクが存在しています。

### 34.1 各ビットマスクについて

PhysicsBody には、以下のビットマスクが用意されています。

- Category mask
- Collision mask
- Contact mask

### 34.2 Category mask

PhysicsBody 用の Category mask となり、ノード同様に AND 演算の結果が 0 以外であれば同じカテゴリーに所属することになります。

### 34.3 Collision mask

こちらのビットマスクは衝突を判別するために使用します。CategoryBitmask で同じカテゴリーに所属していて、互いの CollisionBitmask が AND 演算の結果が 0 以外であれば、衝突判定が行われます。

そのため、CategoryBitmask と CollisionBitmask の要件が合わないと、Dynamic の PhysicsBody は障害物から突き抜けています。

## 34.4 Contact mask

他の PhysicsBody が接触した際に使用する Contact mask で初期値は 0。

値が 0 以外のノード同士がぶつかり合うと、SCNPhysicsContactDelegate に SCNPhysicsContact オブジェクトにメッセージとして送られます。（仕様上、片方が 0 の場合スルーされるはずですが、SCNPhysicsContact が送られる場合があり謎です）

コード上で処理するものですので、Category mask や Collision mask のように衝突処理自体に影響を与えません。

## 34.5 Category mask と Collision mask の使用例

Category mask と Collision mask 使用して、衝突を判別しています。左の画像から球を3つ落下させて、右が物理アニメーション結果です。

手前の球は板で止まり、中央の球は浮いている板で止まり、奥の球は一番下の床もすり抜けてしまいます。

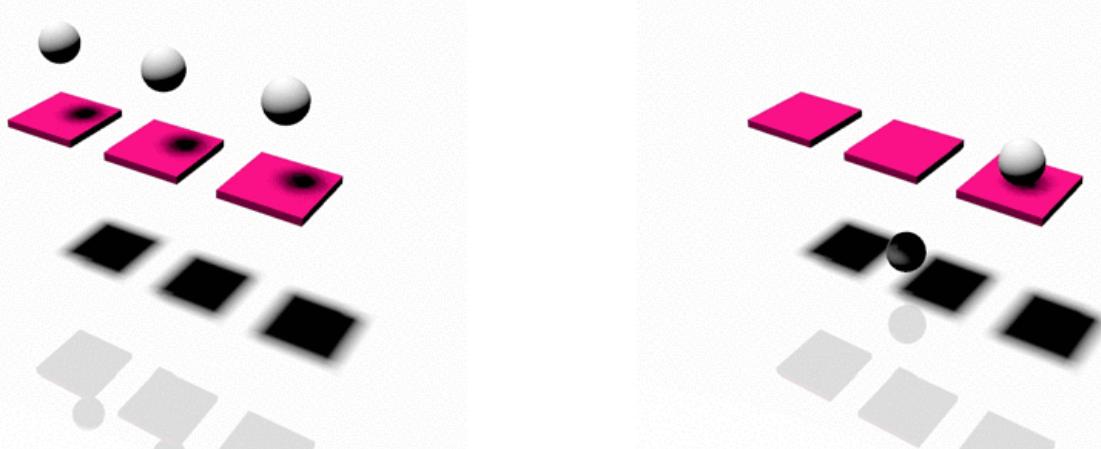


図 34.1 Category mask と Collision mask の使用例

### 34.5.1 球

X 値	-1.5 (奥)	0 (中央)	1.5 (手前)
Category mask	2	1	1
Collision mask	-1	1	-1

### 34.5.2 板

3つとも Static で Category mask と Collision mask は初期値に設定。

- Category mask : 2
- Collision mask -3

### 34.5.3 床 (SCNFloor)

- Category mask : 1
- Collision mask : -3

今までの Bitmask と同様マイナスはすべて許容され、奥の球は Category mask、中央の球は Collision mask の演算で弾かれるため板を通過します。さらに、奥の球は Category mask が床との演算でも弾かれるため、床からも通過してしまいます。

# 第35章

## 物理アニメーションの接触処理

PhysicsWorld 上で、2つ以上の PhysicsBody の接触が起こった場合は SCNPhysicsContactDelegate で PhysicsBody の情報を調べることができます。

### 35.1 注意点

PhysicsBody は contactTestBitMask が 0 以外でないと SCNPhysicsContactDelegate へ情報が送られないので注意が必要です。

また、SCNPhysicsContact は SCNPhysicsContactDelegate の内部で情報が更新されます。そのため、それ以外の場所で SCNPhysicsContact を呼んでも接触の情報は取得できません。

iOS 11 では SCNSceneRendererDelegate を呼ばないと SCNPhysicsContactDelegate が動作しなくなりました。

### 35.2 SCNPhysicsContact で取得できるもの

以下のものが、delegate の contact から取得できるものです。

プロパティ名	説明
nodeA	1 個目のノードを返す
nodeB	2 個目のノードを返す
contactPoint	接地点の座標
contactNormal	接点の法線ベクトル。どの方向から衝突されているかを示す
collisionImpulse	ニュートン秒で表された衝突時の力。弱く衝突しているか、激しく衝突しているかなどを調べることができます
penetrationDistance	互いのノードが重なっている距離

### 35.3 SCNPhysicsContactDelegate の命令

接触した時、接触状態が更新された時、接触が終わった時の3つが用意されています。

- physicsWorld(SCNPhysicsWorld, didBegin: SCNPhysicsContact)
- physicsWorld(SCNPhysicsWorld, didUpdate: SCNPhysicsContact)
- physicsWorld(SCNPhysicsWorld, didEnd: SCNPhysicsContact)

### 35.4 設定手順

1. ViewControllerなどのクラスで SCNPhysicsContactDelegateを呼ぶ。
2. シーンのメソッドで physicsWorld.contactDelegateを設定する
3. 調べたい PhysicsBodyの contactTestBitMaskを0以外にする
4. SCNPhysicsContactDelegateの命令を追加する

### 35.5 SCNPhysicsContactDelegateを設定する

球が落下し床に着くとマテリアルの色が変わり、画面タップで物理アニメーションの位置と色がリセットされます。

本来は physicsWorld(\_ world:, didBegin contact:) 使用時に contact.nodeA や contact.nodeB でノードを調べますが、今回は接触するものが2つしかないため省いています。

リスト 35.1: SCNPhysicsContactDelegateを設定

```
import PlaygroundSupport
import SceneKit

class GameViewController: UIViewController, SCNPhysicsContactDelegate {

    var ballNode:SCNNode!
    var scnView:SCNView!

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
// シーン
let scene = SCNScene()

// contactDelegate を設定
scene.physicsWorld.contactDelegate = self

// カメラ
let cameraNode = SCNNNode()
cameraNode.camera = SCNCamera()
cameraNode.position = SCNVector3(x: 12.68, y: 7.445, z: 12.86)
cameraNode.eulerAngles = SCNVector3(x: ((Float.pi * -22.129) / 180),
y: ((Float.pi * 44.576) / 180), z: 0.0)
scene.rootNode.addChildNode(cameraNode)

// ライト
let lightNode = SCNNNode()
lightNode.light = SCNLight()
lightNode.light!.type = .omni
lightNode.position = SCNVector3(x: 0, y: 12, z: 0)
scene.rootNode.addChildNode(lightNode)

// 球
ballNode = SCNNNode(geometry: SCNSphere(radius: 2))
ballNode.name = "ball"
ballNode.position = SCNVector3(x: 0, y: 5, z: 0)
ballNode.physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)
ballNode.physicsBody?.contactTestBitMask = 1
scene.rootNode.addChildNode(ballNode)

// 床
let floorNode = SCNNNode(geometry: SCNPlane())
floorNode.name = "floor"
floorNode.physicsBody = SCNPhysicsBody(type: .static, shape: nil)
floorNode.physicsBody?.contactTestBitMask = 1
scene.rootNode.addChildNode(floorNode)

// View 設定
scnView = SCNView()
self.view.addSubview(scnView)

// View の Autolayout
scnView.translatesAutoresizingMaskIntoConstraints = false

self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
```

```
"V:[scnView]", options: NSLayoutFormatOptions(rawValue: 0),
metrics: nil, views: ["scnView": scnView]))  
  
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
"H:[scnView]", options: NSLayoutFormatOptions(rawValue: 0),
metrics: nil, views: ["scnView": scnView]))  
  
scnView.scene = scene
scnView.allowsCameraControl = true
scnView.showsStatistics = true
scnView.backgroundColor = UIColor.black  
  
// タップジェスチャー
let tapGesture = UITapGestureRecognizer(target: self, action:
#selector(handleTap(_:)))
scnView.addGestureRecognizer(tapGesture)
}  
  
@objc
func handleTap(_ gestureRecognizer: UIGestureRecognizer) {
    ballNode.geometry?.firstMaterial?.diffuse.contents = UIColor.white
    ballNode.physicsBody?.resetTransform()
}  
  
func physicsWorld(_ world: SCNPhysicsWorld, didBegin contact:
SCNPhysicsContact) {
    // let firstNode = contact.nodeA
    // print("NodeA: \(String(describing: firstNode.name!))")
    ballNode.geometry?.firstMaterial?.diffuse.contents = UIColor.red
}  
  
override var prefersStatusBarHidden: Bool {
    return true
}  
  
override var supportedInterfaceOrientations: UIInterfaceOrientationMask {
    if UIDevice.current.userInterfaceIdiom == .phone {
        return .allButUpsideDown
    } else {
        return .all
    }
}  
  
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
```

```
    }  
  
}  
  
PlaygroundPage.current.liveView = GameViewController()
```

# 第36章

## 物理フィールドの概要

重力、電磁気、乱気流などの外部から与えられる力を領域内に反映させるオブジェクト、`PhysicsField` (物理フィールド) について解説します。

`PhysicsField` は物理アニメーションとパーティクルに適応可能で、コード上からカスタムの `PhysicsField` を作成することができます。

### 36.1 物理フィールドの使用例

例として、次の画像では中央の四角い部分に空気抵抗を模した物理フィールドを設置しており、上から球を同時に落下させ片方が四角に入ります。空気抵抗のあるフィールド入った球は落下速度が遅くなるため、手前の球の方が先に落ちます。

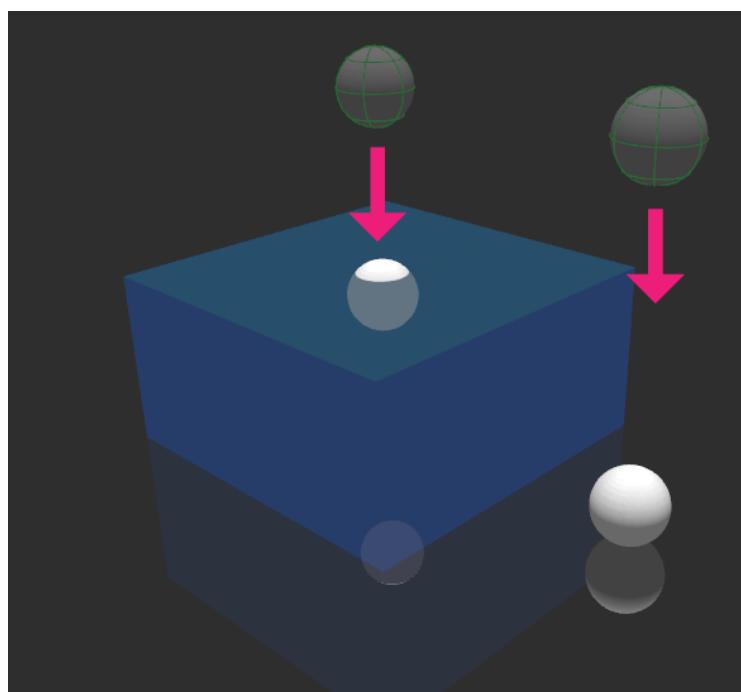


図 36.1 物理フィールドの使用例

## 36.2 SceneKit で用意されている物理フィールド

以下のものは、物理フィールドの種類と共にパラメーターです。

表 36.1 物理フィールドの種類

名称	機能
Drag	動きを減速させる流体摩擦や空気抵抗のフィールド
Vortex	指定した軸を中心に力が循環する渦のフィールド
Radial Gravity	中心に向かって加速する重力フィールド
Linner Gravity	特定の方向に加速する重力フィールド
Noise	ランダムな力を適用するフィールド
Turbulence	オブジェクトの速度に比例した大きさでランダムな力を適用するフィールド
Spring	中心に向かってバネのような力で引っ張るフィールド
Electoric	電荷によって中心から距離に基づいてオブジェクトを引き付けるか、または反発させるフィールド
Magnetic	電荷、速度、フィールド軸の距離に基づいて、オブジェクトを引き付けるか、または反発させるフィールド

表 36.2 共通パラメーター

パラメーター	機能
strength	PhysicsField が適応範囲に与える力の数値
falloffExponent	距離とともに strength の値がどのように減少するかを設定する数
minimumDistance	距離に基づく効果の最小値
isActive	PhysicsField 効果の有効/無効を設定します
isExclusive	PhysicsField が重なっている他のフィールドを無視するかどうかを指定します
halfExtent	PhysicsField の適応範囲を設定する。デフォルト値は無限遠になっています
scope	適応範囲を指定している内側か外側かを決める。halfExtent がデフォルト値の場合は無限遠であるため変更しても変わらない
usesEllipsoidalExtent	デフォルトでは適応範囲は立方体になっていますが、こちらを true にすると球状に変更することができます
categoryBitMask	PhysicsField 用のカテゴリービットマスク。PhysicsBody と PhysicsField ビットマスクの AND 演算が 0 でなければ、PhysicsField を適応します

### 36.2.1 halfExtent の設定

PhysicsField の中心である pivot から x, y, z 座標を指定するとそれに合わせた立方体が適応範囲となります。

PhysicsField を (x: 2.0, y:2.0, z:2.0) に配置した場合で、halfExtent を (x: 0.5, y:0.5, z:0.5) とした設定は、x, y, z で 1.5 ~ 2.5 が有効範囲となります。

### 36.2.2 scope の outside

デフォルト値は内側の設定になっているので問題ないのですが、outside にした場合、設定した適応範囲の外側すべてになるため注意が必要です。

### 36.2.3 フィールド効果の減衰

フィールド効果の減衰は  $\text{pow}(\text{distance} - \text{minRadius}, -\text{falloff})$  で計算されています。falloff-Exponent の数が 0 より大きい場合、フィールドの効果は近くの PhysicsBody が強くなり、デフォルトの falloffExponent の値は、フィールドタイプによって異なります。

# 第37章

## 各物理フィールドについて

---

9個ある物理フィールドを説明していきます。

### 37.1 Drag Field

動きを減速させる流体摩擦や空気抵抗のフィールド。

#### 37.1.1 Strength

適応範囲での抵抗の力になります。

### 37.2 Vortex Field

指定した軸を中心に力が循環する渦のフィールド。

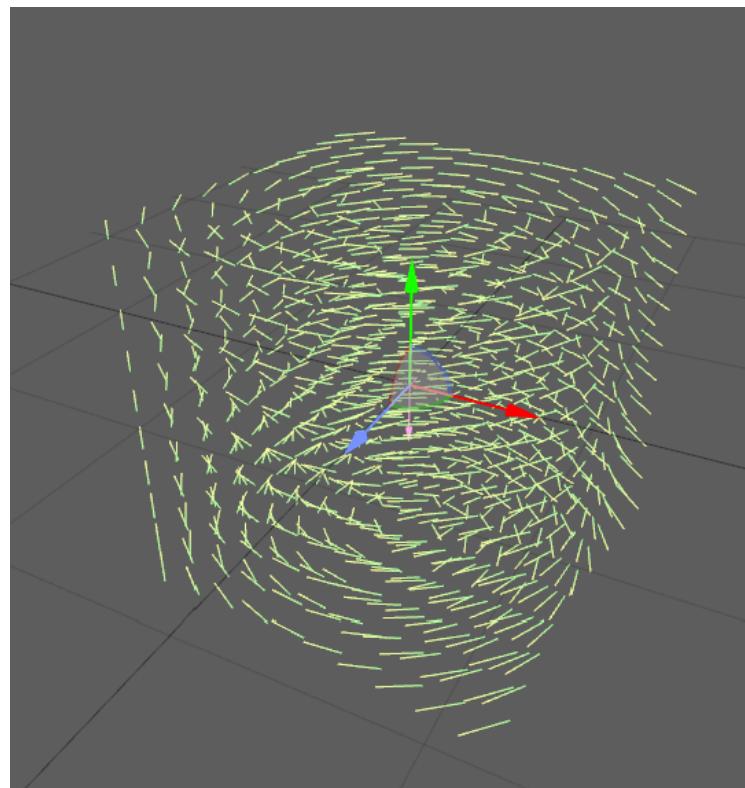


図 37.1 Scene Editor での表示

### 37.2.1 Strength

中心から渦状に外へ弾き出す力。

### 37.2.2 Direction (SCNVector3)

回転軸の方向。デフォルト値は (x:0, y:-1, z:0) で Y 軸方向に反時計回りで回ります。

### 37.2.3 Offset (SCNVector3)

回転軸の中央位置を変更します。

## 37.3 Radial Gravity Field

中心に向かって加速する重力フィールド。

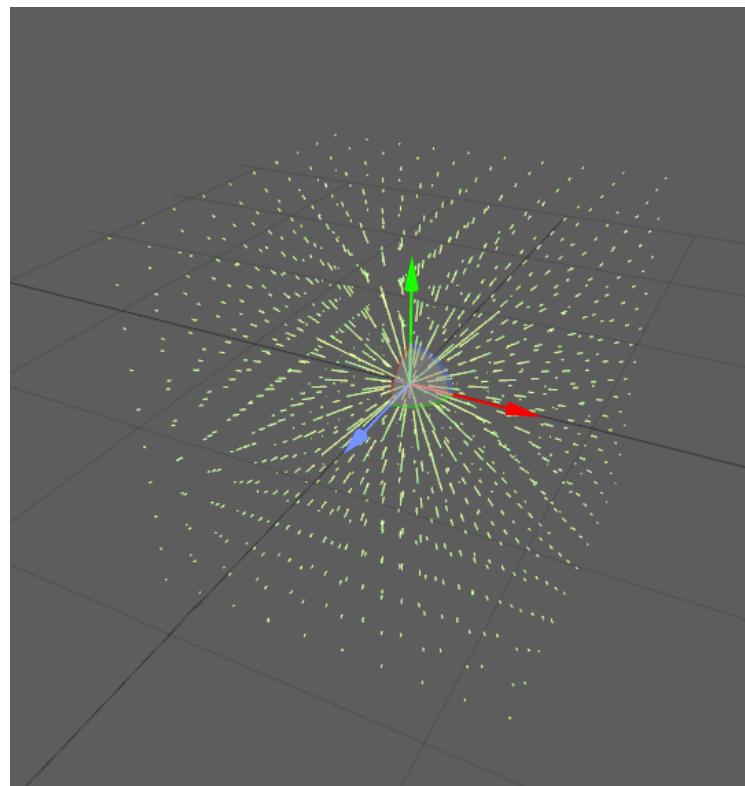


図 37.2 Scene Editor での表示

### 37.3.1 Strength

メートル/秒でフィールドの中心へ加速させる力でマイナスの場合は押し出します。重力のフィールドであるため質量が考慮されます。

### 37.3.2 Offset (SCNVector3)

効果が発生する中心の位置を変更します。

## 37.4 Linner Gravity Field

特定の方向に加速する重力フィールド。

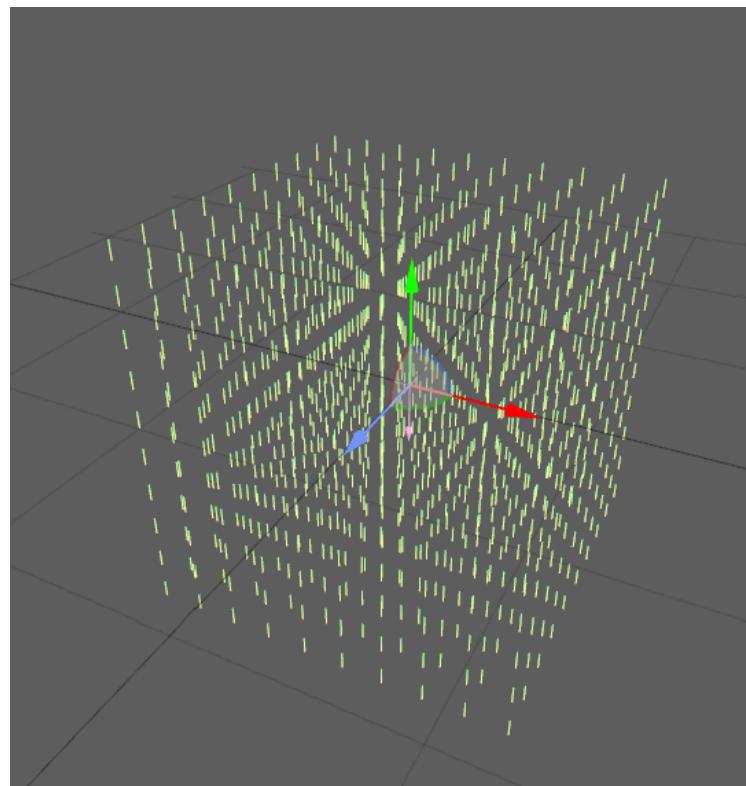


図 37.3 Scene Editor での表示

### 37.4.1 Strength

指定した方向にメートル/秒で加速し、重力のフィールドあるため質量が考慮されます。

### 37.4.2 Direction (SCNVector3)

重力が発生する方向で、初期値の X、Z は 0 で、Y 軸方向は -1 になっており下向きになっています。

## 37.5 Noise Field

ランダムな力を適用するフィールド。

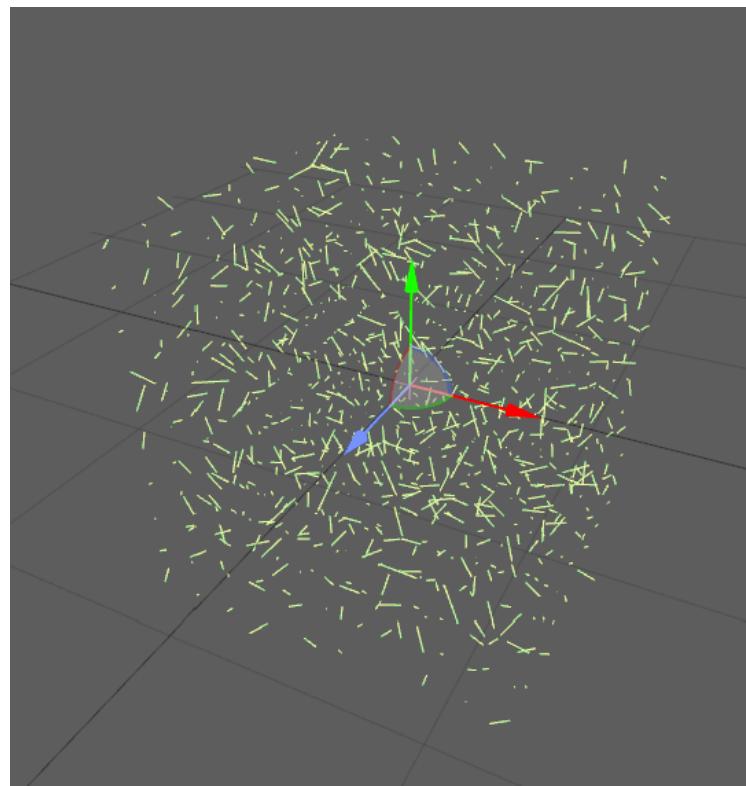


図 37.4 Scene Editor での表示

### 37.5.1 Strength

Simplex noise のアルゴリズムを使用したノイズの大きさです。

### 37.5.2 smoothness (CGFloat)

ノイズの滑らかさ。0.0 が最大で、1.0 でなくなる

### 37.5.3 speed (CGFloat)

時間でのノイズの変化。0.0 で変化しなくなる

## 37.6 Turbulence Field

オブジェクトの速度に比例した大きさでランダムな力を適用するフィールド。

### 37.6.1 Strength

Noise と同じような乱気流効果の大きさで、Noise より速く激しく揺れるような振る舞いになります。

### 37.6.2 smoothness (CGFloat)

ノイズの滑らかさ。0.0 が最大で、1.0 でなくなります。

### 37.6.3 speed (CGFloat)

時間でのノイズの変化。0.0 で変化しなくなります。

## 37.7 Spring Field

中心に向かってバネのような力で引っ張るフィールド。

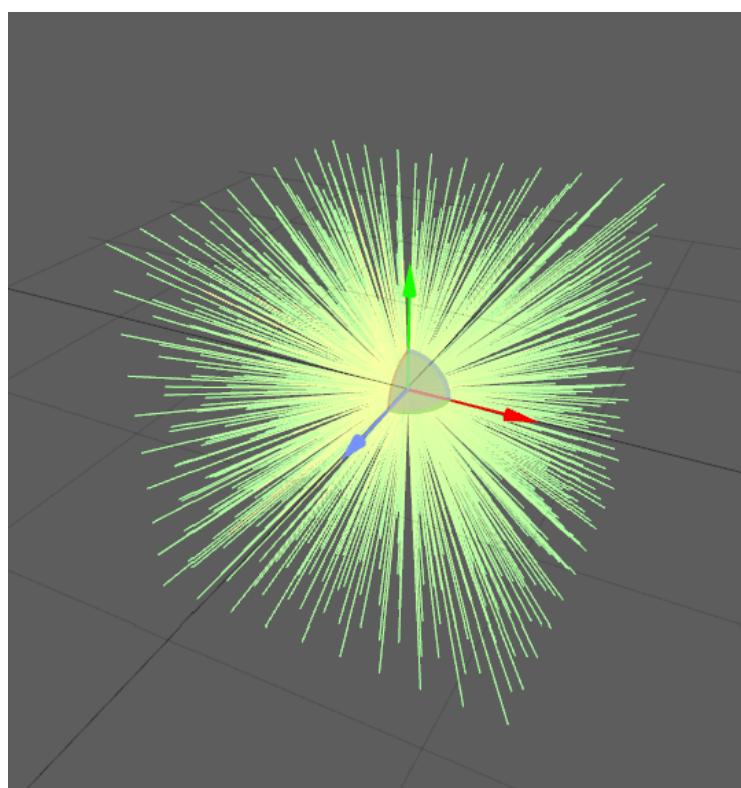


図 37.5 Scene Editor での表示

### 37.7.1 Strength

バネの力の強さ。値を高くすると速く中央に戻ります。

### 37.7.2 Offset (SCNVector3)

効果が発生する中心の位置を変更します。

## 37.8 Electric Field

電荷によって中心から距離に基づいてオブジェクトを引き付けるか、または反発させるフィールド。

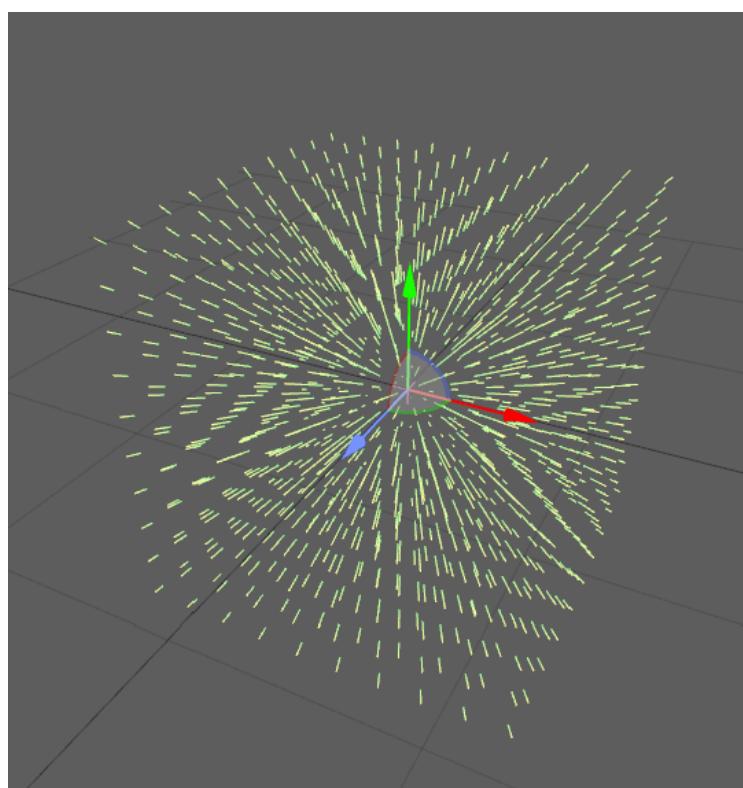


図 37.6 Scene Editor での表示

### 37.8.1 Strength

電荷によって引き付ける（遠ざける）力。こちらは中央に引き付けます。（PhysicsBody が負の電荷を持つ場合は逆になります）

### 37.8.2 Offset (SCNVector3)

磁界の中央位置を変更します。

## 37.9 Magnetic Field

電荷、速度、フィールド軸の距離に基づいて、オブジェクトを引き付けるか、または反発させるフィールド。

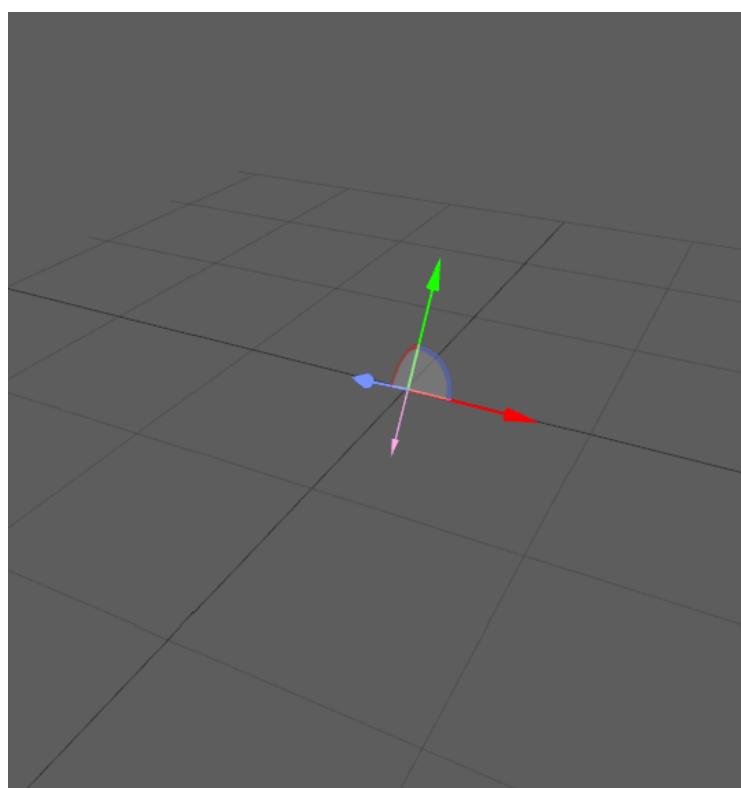


図 37.7 Scene Editor での表示

### 37.9.1 Strength

Electric Field と同様になり、こちらは直線電流の磁場のようなフィールドであるため、プラス値で半時計回り、マイナス値で時計周りになります（PhysicsBody が負の電荷を持つ場合は逆）

## 37.10 パーティクルを物理フィールドでも動くように設定する

初期状態のパーティクルではオフになっているので、SCNParticle の isAffectedByPhysicsFields を true にしてください。

## 37.11 設定例

リスト 36.1: 設定例

```
let field = SCNPhysicsField.vortex()

field.isActive = true
field.isExclusive = false

field.strength = 1
field.falloffExponent = 0
field.minimumDistance = 0

field.halfExtent = SCNVector3(10,10,10)
field.usesEllipsoidalExtent = false
field.scope = .insideExtent

field.direction = SCNVector3(0,0,0)
field.offset = SCNVector3(0,0,0)

field.categoryBitMask = -1

let fieldNode = SCNNNode()
fieldNode.physicsField = field

scene.rootNode.addChildNode(fieldNode)
```



# 第38章

## パーティクルシステム

パーティクルシステムとは画像を使用し、粒状のものへ振る舞いの設定を行い、煙、雨、紙吹雪、花火の効果を表現します。

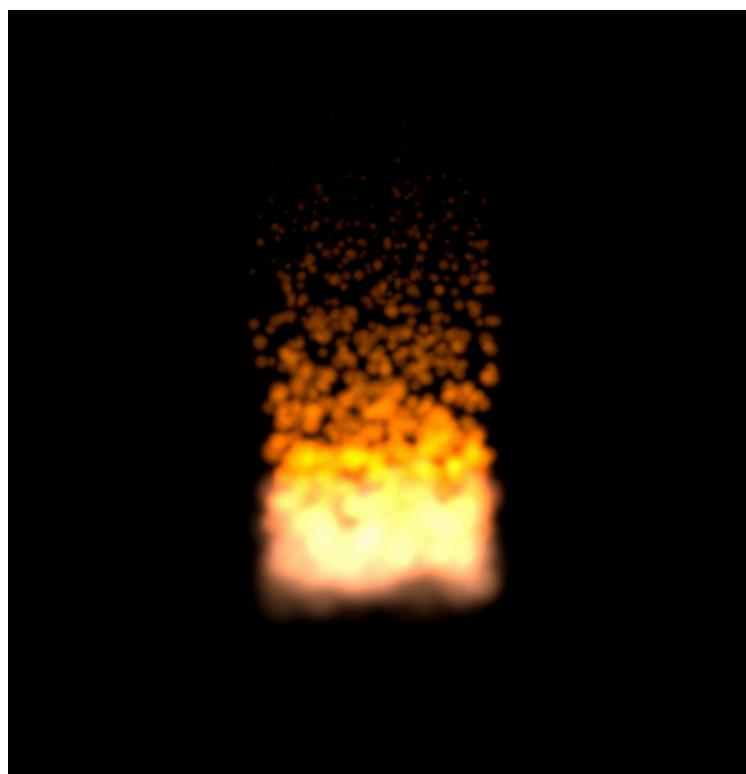


図 38.1 炎を模したパーティクル

その粒子の一つ一つをパーティクルと呼び、パーティクルのレンダリングは SceneKit で存在するオブジェクトで唯一 Scene Graph と切り離され存在となっています。数万と描画される可能性があるため、内部的に処理しているのだと思われます。

負荷の問題を考えるとあまり現実的ではないので、あまり使用しないと思われますが、他のゲームエンジンと異なりパーティクルにジオメトリを設定することはできません。自力で物理アニメーションを行う必要があります。

パーティクルの設定値が多く、Swift Playgrounds で設定するには項目が多すぎるため本書では軽い説明に留めます。

### 38.1 SCNParticleSystem の設定方法の種類

- 全てコードで作成する
- SceneKit Particle System ファイル (scnp) を作成し、コードから利用する
- Scene ファイル (scn) の Scene Graph で Particle System を作成する

設定項目が多く動作の状況が掴みづらいので、大体は scn か scnp から利用することが多くなると思います。

### 38.2 SceneKit Particle System ファイル (scnp) をシーンに適応する

SCNParticleSystem からファイルを読み込み、ノードの addParticleSystem から適応させるだけです。事前に Xcode 上で scnp を作成しておけば Swift Playground でも割と簡単にパーティクルの表現を試すことができます。

リスト 37.1: scnp ファイルをシーンに適応する

```
let particle = SCNParticleSystem(named: "Fire.scnp", inDirectory: "")!
let particleNode = SCNNNode()
particleNode.addParticleSystem(particle)
scene.rootNode.addChildNode(particleNode)
```

Fire.scnp は直線からパーティクルが放出されていますがジオメトリをエミッター（発射口）にすることで様々な形でパーティクルを放出することができます。

リスト 37.2: 球をエミッター

```
let particle = SCNParticleSystem(named: "Fire.scnp", inDirectory: "")!  
  
let particleNode = SCNNNode()  
particleNode.addParticleSystem(particle)  
  
scene.rootNode.addChildNode(particleNode)
```

### 38.3 パーティクルシステムの簡単な流れ

#### 38.3.1 パーティクルの画像設定

各パーティクルにテクスチャ画像を適応し、色合い、ブレンディングモードなどのパラメータを調整し外観を設定します。また、アニメーションする画像として複数枚設定することで、鳥の群れや多段階で発生する爆発など表現することができます。

#### 38.3.2 パーティクルの寿命

パーティクルは発射口となるエミッタの始点から、消えるまで位置や色を変更することでアニメーションを表現します。

各パーティクルのが消えるまでを birthRate と particleLifeSpan とし、掛け合わしたもの寿命として設定しますが、寿命を長くしすぎると画面上のパーティクルが多くなりすぎ消費電力と演算に時間がかかるため注意が必要です。

#### 38.3.3 エミッタ

発射口となる部分で、单一点もしくはジオメトリをエミッタに設定することができます。emissionDuration の設定次第で一定期間アイドルさせたりすることができます。

#### 38.3.4 バリエーション

パーティクルシステムプロパティには、ランダムに変化させるプロパティがあります。例えば、大きさをランダム化し大きさに幅を持たせることができます。

### 38.3.5 動き

パーティクルは方向、速度、角速度、加速度などのシンプルな物理シミュレーションによって移動が行われ、設定している寿命がきて消滅するまでアニメーションを行います。パーティクルは物理シミュレーションであるため PhysicsWorld のジオメトリによる障害物判定や PhysicsField を使うことで複雑な効果をアニメーションで行うことができます。

### 38.3.6 他の機能

SCNParticlePropertyController を使用すると CoreAnimation の機能から個々のパーティクルの振る舞いを調整しエフェクトを作成すことができ、systemSpawnedOnCollision を使用することでパーティクルが障害物に衝突した際に処理ができます。新たなパーティクルが生成される設定など様々な振る舞いを行うことができます。

# 第39章

## コンストRAINT (制約)

---

3DCG では移動や回転、拡大縮小がある一定まで操作させそれ以上動かさないよう制約をかけることができます。

いくつか SDK で設定が用意されていますが、自分でコンストRAINTを作成することもできます。

### 39.1 コンストRAINTの種類

カスタム以外で用意されているものは以下のものです。本書ではスキニングを紹介していないため、Inverse Kinematics については省きます。

- SCNBillboardConstraint
- SCNLookAtConstraint
- SCNDistanceConstraint (iOS 11 で追加)
- SCNAvoidOccluderConstraint (iOS 11 で追加)
- SCNAccelerationConstraint (iOS 11 で追加)
- SCNSliderConstraint (iOS 11 で追加)
- SCNReplicatorConstraint (iOS 11 で追加)

### 39.2 コンストRAINTの使用例

ノードの持つ constraints に配列で渡すだけです。配列で設定するため、コンストRAINT の効果を複数設定することが可能です。

リスト 38.1: constraints

```
sunHaloNode.constraints = [SCNBillboardConstraint()]
```

以降、各機能についての説明

### 39.3 SCNBillboardConstraint

ノードの Z 軸プラス方向正面とし、カメラに対して必ず正面を向かせる回転のコンストRAINTです。

freeAxis を設定することで、全ての軸で回転するか、X,Y,Z 方向を固定することができます。

#### 39.3.1 freeAxis の設定値

- X
- Y
- Z
- all

### 39.4 SCNLlookAtConstraint

ノードを設定した対象のノードへ振り向くようにさせる回転のコンストRAINTです。

target で目標を決めて、isGimbalLockEnabled は roll 軸 (X 軸) の回転を制限します。

リスト 38.2: SCNLlookAtConstraint

```
let constraint = SCNLlookAtConstraint(target: targetNode)
constraint.isGimbalLockEnabled = true

node.constraints = [constraint]
```

## 39.5 SCNDistanceConstraint (iOS 11 で追加)

設定した球状の範囲から対象のノードが離れると、ノードの方へついて行こうとする位置のコンストRAINTです。

`minimumDistance` で自身の中心から `maximumDistance` まで範囲をつくりそのエリアから対象が離れると対象についていくように動きます。

キャラクターがある程度移動した際にカメラがキャラクターを追ったり、キャラクターの移動でついてくるサブキャラクターなどに設定すると便利です。

リスト 38.3: SCNDistanceConstraint

```
let constraint = SCNLookAtConstraint(target: targetNode)
constraint.minimumDistance = 0
constraint.maximumDistance = 10

node.constraints = [constraint]
```

## 39.6 SCNAvoidOccluderConstraint (iOS 11 で追加)

遮蔽物となる `target` を設定するとそのジオメトリ内側から出ることができなり進行を妨げる位置のコンストRAINTです。`bias` を増やすと遮蔽するジオメトリより前で遮蔽され、`delegate` で接触前と接触後を設定できます。

ゲーム空間でキャラクターが移動可能範囲を設定するときなどに使用すると便利です。

リスト 38.4: SCNAvoidOccluderConstraint

```
let constraint = SCNAvoidOccluderConstraint(target: targetNode)
constraint.bias = 0
constraint.occluderCategoryBitMask = 0
constraint.delegate = self as! SCNAvoidOccluderConstraintDelegate

node.constraints = [constraint]
```

## 39.7 SCNAccelerationConstraint (iOS 11 で追加)

座標移動した際に、すぐその場所に移動せず徐々に座標まで移動する動きのコンストRAINTです。SCNDistanceConstraint や移動のコンストRAINTと併用することが多いと思われます。

damping は空気抵抗、decelerationDistance は減速する距離、maximumLinearAcceleration は最大の加速、maximumLinearVelocity 最大の速度です。

リスト 38.5: SCNAccelerationConstraint

```
let constraint = SCNAccelerationConstraint()
constraint.damping = 0.1
constraint.decelerationDistance = 1
constraint.maximumLinearAcceleration = 200
constraint.maximumLinearVelocity = 200

node.constraints = [constraint]
```

## 39.8 SCNSliderConstraint (iOS 11 で追加)

設定するとシーン上のジオメトリを避けて移動しようとする動きのコンストRAINTです。

radius 自身の当たり判定の半径、offset 障害物となる対象の当たり判定のオフセット値隣、カテゴリビットマスクを使用して適応するジオメトリ決めることができます。

リスト 38.6: SCNSliderConstraint

```
let constraint = SCNSliderConstraint()
constraint.collisionCategoryBitMask = 0
constraint.offset = SCNVector3(0, 0, 0)
constraint.radius = 2.0

node.constraints = [constraint]
```

## 39.9 SCNReplicatorConstraint (iOS 11 で追加)

対象のノードの位置、回転、拡大縮小の値の全て、もしくは一部をコピーし自身に適応します。要するに対象と同じ状態になります。

以下を true で有効化し、各 Offset を変更する

- replicatesOrientation
- replicatesPosition
- replicatesScale

リスト 38.7: SCNReplicatorConstraint

```
let constraint = SCNReplicatorConstraint(target: targetNode)

constraint.replicatesOrientation = true
constraint.replicatesPosition = true
constraint.replicatesScale = false

constraint.orientationOffset: SCNVector4(0, 0, 0, 1)
constraint.positionOffset: SCNVector3(0, 0, 0)
constraint.scaleOffset: SCNVector3(0, 0, 0)

node.constraints = [constraint]
```

# 第40章

## SpriteKit のシーンをテクスチャとして使用する

---

SceneKit では、2DCG フレームワークである SpriteKit のシーンや SKTexture をマテリアルプロパティの `contents` に渡すことができます。使用前に他のフレームワークを使用するため「import SpriteKit」を記述する必要があります。

SpriteKit の解説は本書ではありませんが、Web サイトや何冊か出版されていますのそちらを参考にしてみてください。

### 40.1 SKTexture をテクスチャとして使用する

`UIImage` の代わりに `SKTexture` を使用することができます。

リスト 39.2: `SKTexture` を適応する

```
let skTexture = SKTexture(imageNamed: "texture.png")
boxTexture.geometry?.firstMaterial?.diffuse.contents = skTexture
```

また、`generatingNormalMap()` を使用して Normal マップをコードから作成することができます。

リスト 39.2: `SKTexture` を適応する

```
let skTexture = SKTexture(imageNamed: "texture.png")
boxNode.geometry?.firstMaterial?.diffuse.contents = skTexture
boxNode.geometry?.firstMaterial?.normal.contents =
skTexture.generatingNormalMap()
```

## 40.2 SpriteKit のシーンをテクスチャとして使用する

SKTexture と同様に、SKScene のそのまま contents に渡すことができます。

### 40.2.1 SpriteKit の SKVideoNode を使用し動画のテクスチャを使用する

AVFoundation の AVPlayer に再生する動画を設定した AVPlayerItem 設定し、SKVideoNode に設定した AVPlayer を渡し、SKScene に追加します。

リスト 39.3: SKVideoNode で動画を再生する

```
import PlaygroundSupport
import UIKit
import SceneKit
import SpriteKit
import AVFoundation

class GameViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let scene = SCNScene()

        let videoW = 720.0
        let videoH = 404.0

        let cameraNode = SCNNNode()
        cameraNode.camera = SCNCamera()
        cameraNode.position = SCNVector3(x: 0, y: 0, z: 0)
        scene.rootNode.addChildNode(cameraNode)

        cameraNode.camera?.vignettingPower
        cameraNode.camera?.vignettingIntensity

        let lightNode = SCNNNode()
        lightNode.light = SCNLight()
        lightNode.light!.type = .omni
        lightNode.position = SCNVector3(x: 0, y: 10, z: 10)
        scene.rootNode.addChildNode(lightNode)
```

```
let ambientLightNode = SCNNode()
ambientLightNode.light = SCNLight()
ambientLightNode.light!.type = .ambient
ambientLightNode.light!.color = UIColor.darkGray
scene.rootNode.addChildNode(ambientLightNode)

// AVPlayer の設定
let item = AVPlayerItem(url: URL(fileURLWithPath:
Bundle.main.path(forResource: "movie", ofType: "mp4")!))
let videoPlayer = AVPlayer(playerItem: item)

videoPlayer.actionAtItemEnd = AVPlayerActionAtItemEnd.none;
NotificationCenter.default.addObserver(self,
selector: #selector(self.stateEnd),
name: NSNotification.Name("AVPlayerItemDidPlayToEndTimeNotification"),
object: videoPlayer.currentItem)

videoPlayer.play()

// SKScene の設定
let skScene = SKScene()
skScene.backgroundColor = UIColor.black
skScene.size = CGSize(width: videoW, height: videoH)

let skVideoNode = SKVideoNode(avPlayer: videoPlayer)
skVideoNode.size = CGSize(width: videoW, height: videoH)
skVideoNode.position = CGPoint(x: videoW / 2, y: videoH / 2)
skVideoNode.yScale = -1.0
skScene.addChild(skVideoNode)

// SCNPlane の設定
let planeNode = SCNNode(geometry: SCNPlane(width:
CGFloat(videoW/100), height: CGFloat(videoH/100)))
planeNode.geometry?.firstMaterial?.diffuse.contents = skScene
planeNode.position = SCNVector3(0, 0, -12)
scene.rootNode.addChildNode(planeNode)

// View 設定
let scnView = SCNView()
self.view.addSubview(scnView)

// View の Autolayout
scnView.translatesAutoresizingMaskIntoConstraints = false
self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
"V:[scnView] |", options: NSLayoutConstraint.Options(rawValue: 0),
```

```
        metrics: nil, views: ["scnView": scnView]))  
    self.view.addConstraints(NSLayoutConstraint.constraints(withVisualFormat:  
        "H:[scnView]!", options: NSLayoutFormatOptions(rawValue: 0),  
        metrics: nil, views: ["scnView": scnView]))  
  
    scnView.scene = scene  
    scnView.allowsCameraControl = true  
    scnView.showsStatistics = true  
    scnView.backgroundColor = UIColor.black  
}  
  
// ループ再生用  
@objc func stateEnd(notification: NSNotification) {  
    let avPlayerItem = notification.object as? AVPlayerItem  
    avPlayerItem?.seek(to: kCMTimeZero, completionHandler: nil)  
}  
  
override var shouldAutorotate: Bool {  
    return true  
}  
  
override var prefersStatusBarHidden: Bool {  
    return true  
}  
  
override var supportedInterfaceOrientations: UIInterfaceOrientationMask {  
    if UIDevice.current.userInterfaceIdiom == .phone {  
        return .allButUpsideDown  
    } else {  
        return .all  
    }  
}  
  
override func didReceiveMemoryWarning() {  
    super.didReceiveMemoryWarning()  
    // Release any cached data, images, etc that aren't in use.  
}  
}  
  
PlaygroundPage.current.liveView = GameController()
```

# 第41章

# HDR

---

HDR (High Dynamic Range) はすでに Apple 製品使用されていますが、ざっくり説明すると、表示される画面は色の情報しか持たないが、ピクセルごとに色とは別に光の情報を持つものです。現実の色のより広いレンジを持ち、画像処理の際に多彩な表現を行うことができます。

端末のディスプレイ上では HDR ではなく LDR (Low Dynamic Range) のものがまだ多く、光の情報を持たないため、トーンマッピングという手法を使い HDR の色を落とし込む画像処理を行います。

代表的な使用例ですと画面の露出表現などができます。

## 41.1 HDR の使用方法

SCNCamera のプロパティ `wantsHDR` を `true` にし、以下の設定を調整します。

- Average Gray
- White Point
- Auto Adapt
- Speed Factor (Brightening, Darkening)
- Exposure (Minimum, Maximum, Offset)
- Bloom (Intensity, Threshold, Blur Radius)

## 41.2 Average Gray

HDR を使用する際、Photoshop のトーンカーブのように光の情報をどのように LDR に適応するかトーンマッピングカーブを作成します。

Average Gray はトーンマッピングカーブの中間点として使用する輝度レベルを設定し、暗

いまたは明るいコンテンツのシーンを調整することができます。

値を高くすると画面全体の輝度高くなり、 値を低くすると輝度低くなります。

### 41.3 White Point

ハイライトとシャドウ間で光の情報の緩やかな遷移、 または急な遷移を行います。 デフォルト値は 1.0

### 41.4 Adaptation

#### 41.4.1 Auto Adapt

チェックを入れると、 現在のカメラが映す輝度を自動的に測定し、 それに応じて露出レベルを調整されます。 さらに、 遮蔽物などでシーンの輝度が変化すると、 自動的に新しい露出レベルへ変更するアニメーションが行われます。

ドキュメントではデフォルトでオンとなっていますが、 値は `false` が返ります。

#### 41.4.2 Speed Factor

`Brightening` と `Darkening` の値があり、 暗い領域から明るくなる領域、 または明るいところから暗くなる際の持続時間になります。

デフォルト値では `Brightening` は 0.4、 `Darkening` 0.5 と変化が異なります。

### 41.5 Exposure

#### 41.5.1 Minimum、 Maximum

露光量と最小値を最大値で、 最小値を下げるとき暗い部分が露出不足 (均一に黒) になり、 最大値をあげると露出オーバー (均一に白) になります。

1.0 で 2 倍、 2.0 で 4 倍、 -1.0 で 1/2。 差が大きいほど、 コントラストが低くなります。

初期値は `Minimum` が -15、 `Maximum` が 15。

### 41.5.2 Offset

トーンマッピングカーブの明暗にオフセット（偏り）を設定し、初期値は 0.0。プラス値を与えると画面全体の輝度が上がり、マイナス値を与えると下がります。

## 41.6 Bloom

シーンのハイライト（明るい色の領域）に柔らかいボケを追加し、明るいハイライトを実際の人間の目や物理カメラに表示するような方法をシミュレートするブルームエフェクトを与えます。

輝度によって処理が行われるため、設定によってはエフェクトが表示されないことがあるので注意してください。

### 41.6.1 Intensity

ブルームエフェクトの強さを設定します。値を小さくすると微妙な効果を得られ、大きくすると非常に明るい輝きが発生します。

初期値は 0.0 で効果は現れません。

### 41.6.2 Threshold

ブルームエフェクトを発動するために必要な輝度レベルを設定します。初期値は 1.0 で値を小さくするとシーンの広範囲に適応され、値を大きくするとハイライトのみにエフェクトが適応されます。

### 41.6.3 Blur Radius

ブルームエフェクトでハイライトに適用され、ソフトグローのぼかしのピクセル単位の半径を設定します。

値を大きくすると広範囲で柔らかいグローが表現され、0 で無効になり、初期値は 4.0 ピクセルです。

## 41.7 サンプルコード

画面に配置するオブジェクトによって、絵作り上、各パラメーターの調整が必要になるため、Xcode の Scene Editor のシーンファイルで調整した方がよいかと思われます。

リスト 40.1: HDR

```
let camera = SCN Camera()

// HDR
camera.wantsHDR = true
camera.averageGray = 0.18
camera.whitePoint = 1.0

// Adaptation
camera.wantsExposureAdaptation = true
camera.exposureAdaptationBrighteningSpeedFactor = 0.4
camera.exposureAdaptationDarkeningSpeedFactor = 0.6

// Exposure
camera.minimumExposure = -15.0
camera.maximumExposure = 15.0
camera.exposureOffset = 0.0

// Bloom
camera.bloomIntensity = 0.0
camera.bloomThreshold = 0.5
camera.bloomBlurRadius = 4.0

let cameraNode = SCNNNode()
cameraNode.camera = camera
```

# 第42章

## ポストプロセス

---

カメラとなる SCNCamara では画面描画後に簡易的な画像処理を行うことが可能で、使用できる処理は以下のものです。本書では紹介しませんが、さらに複雑な処理をしたい場合は SCNTechnique を使用し、自前でシェーダーを書いて合成します。

- 画面の周りを暗くするケラレ処理を行うビンテージ
- ブラウン管のような色のにじみを出すカラーフリンジ
- 簡易的な色の調整を行うカラーグレーディング
- カメラ撮影時に起こる残像現象であるモーションブラー
- スクリーンスペースアンビエントオクルージョン

スクリーンスペースアンビエントオクルージョンは別の章で紹介します。

### 42.1 Vignetting

画面にケラレ処理を行います。Power が適応範囲で、Intensity が適応度合いで、Power が 0 の場合は効果が現れません。

リスト 41.1: Vignetting

```
cameraNode.camera?.vignettingPower = 1  
cameraNode.camera?.vignettingIntensity = 1
```



図 42.1 Vignetting: Power = 1, Intensity = 1

## 42.2 Color Fringe

フリンジの処理を行います。Strength の値を大きくするほどフリンジの色のシフトと広がりが顕著になり、Intensity が適応度合いなり、Strength が 0 の場合は効果が現れません。

### リスト 41.2: Color Fringe

```
cameraNode.camera?.colorFringeStrength = 1  
cameraNode.camera?.colorFringeIntensity = 1
```



図 42.2 Vignetting: Strength = 4, Intensity = 1

## 42.3 Saturation

画面の彩度を設定します。初期値は 1.0 で 0 になるとグレイスケールになります。

リスト 41.3: Saturation

```
cameraNode.camera?.saturation = 1.0
```



図 42.3 Saturation: 1.0

## 42.4 Contrast

画面のコントラスト変更します。初期値は 0.0。

リスト 41.5: Contrast

```
cameraNode.camera?.contrast = 4.0
```



図 42.4 Contrast: 4

## 42.5 モーションブラー

実際のカメラでは速い動きを物体を撮る際、カメラ性能にもよりますが残像が出る場合があり、それを SceneKit のカメラである SCNCamera でシミュレーション行います。

設定する値は motionBlurIntensity しかありません。初期値は 0 で大きくした場合に効果が顕著になります。

リスト 41.5: Contrast

```
cameraNode.camera?.motionBlurIntensity = 4.0
```

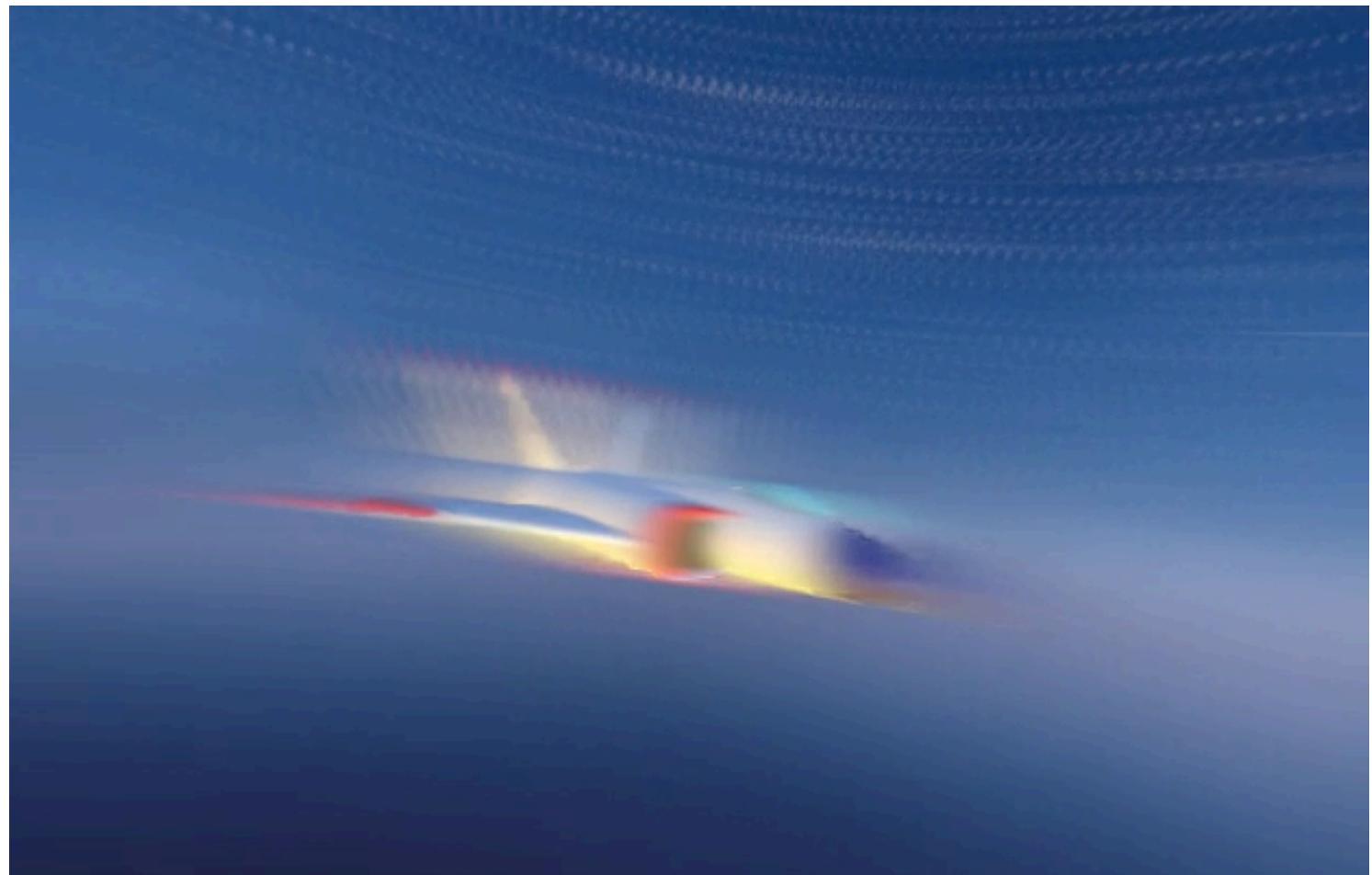


図 42.5 Contrast: 4

# 第43章

## 被写界深度

---

カメラとなる SCNCamara で被写界深度のシミュレーションを行います。iPhone のカメラのポートレートモードのような、実際のカメラと同様に距離でオブジェクトにピントを合わせて、他がボケるさせる効果の設定を行います。

### 43.1 概要

Xcode 8 で使用していた値は現状すべてのプロパティが廃止され名称が変わりました。設定できる値は以下のものです。

- wantsDepthOfField
- focusDistance
- fStop
- apertureBladeCount
- focalBlurSampleCount

wantsDepthOfField を true にして、主に焦点距離 (focusDistance) と F 値 (fStop) で調整を行います。

apertureBladeCount はボケた時に出る絞りの角の数で、focalBlurSampleCount はブラー処理の回数で数が大きくなれば滑らかになりますが処理が重くなります。

### 43.2 焦点距離と F 値

下の画像では、左がレンズ側で対象物を取った時の軽い解説図になります。左右の縦幅のが大きいほどボケる率が上がります。左のような焦点距離を近い場合は範囲が増えるため、対象物以降急激にボケます。逆に右のように遠い場合、あまり周りはボケなくなります。

また、このような構造になっているため、F 値の値を小さくするとさらにボケる割合が上が

ります。

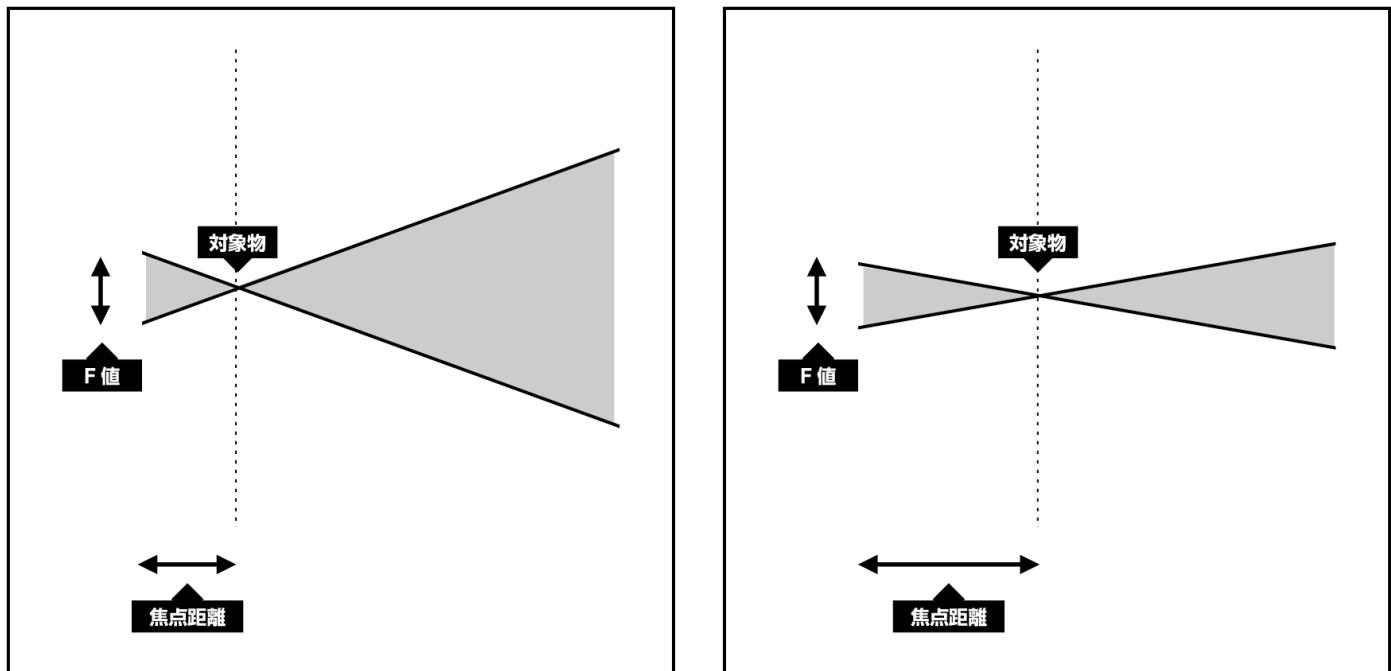


図 43.1 焦点距離と F 値

わかりづらいのですが、次の画像は極めて近い焦点距離と遠い焦点距離使用したものです。近いものはほぼ画面全体がボケてしまっています。



図 43.2 焦点距離 1m



図 43.3 焦点距離 10m

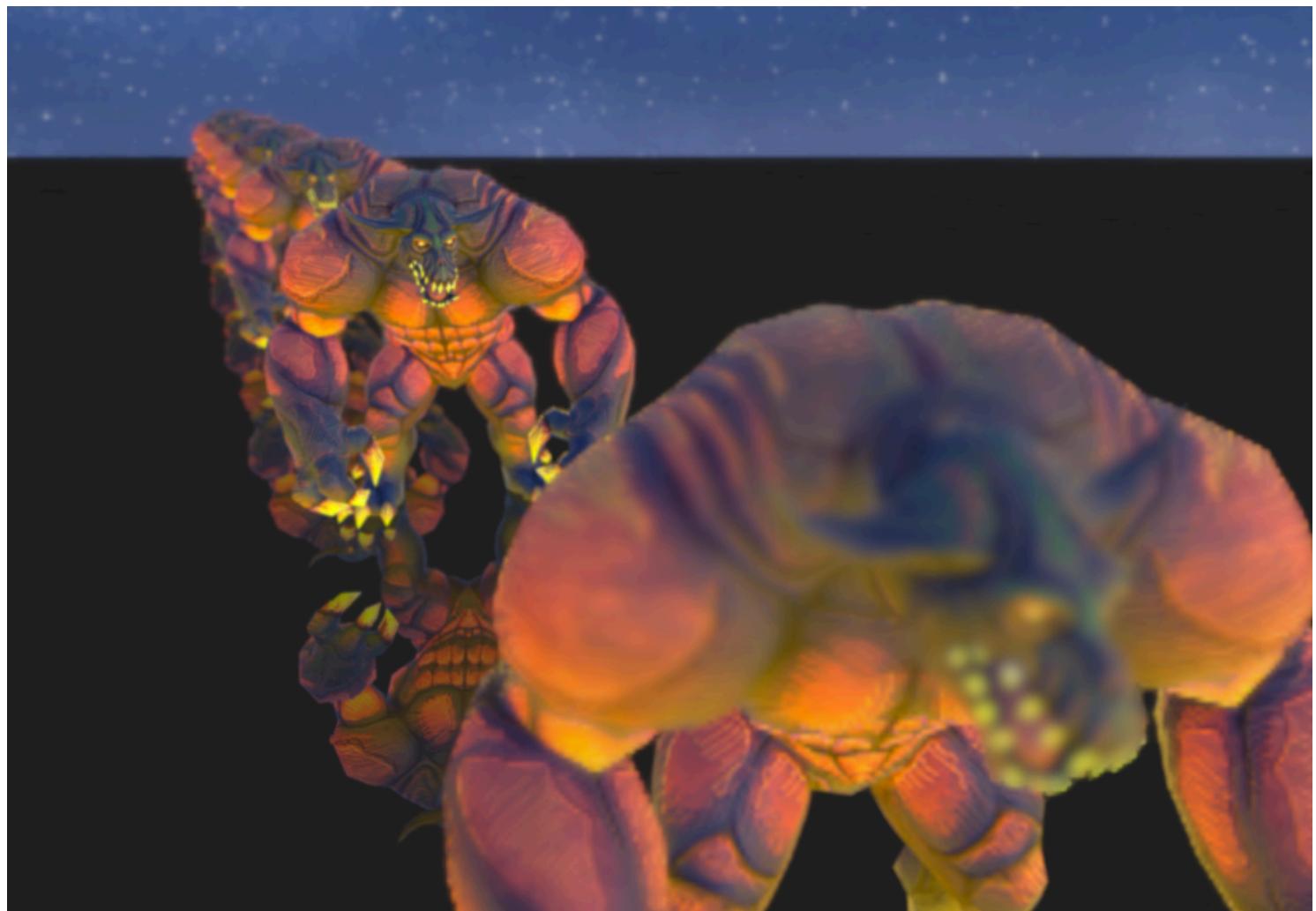


図 43.4 焦点距離を対象に合わせたもの

### 43.3 使用例

リスト 42.1: 被写界深度

```
let camera = SCN Camera()

camera.wantsDepthOfField = true
camera.focusDistance = 1
camera.fStop = 0.154
camera.apertureBladeCount = 5
camera.focalBlurSampleCount = 25

let cameraNode = SCNNNode()
```

```
cameraNode.camera = camera
```

# 第44章

## フォグ（霧）

シーン全体に適応される効果で霧がかかった効果を表現します。ゲームなど遠景のオブジェクトを全て描画してしまうと処理が重くなってしまうためこのような処理でごまかします。

### 44.1 フォグの効果の原理

白く霧がかかった場所では、遠ざかるものはどんどん白くなって背景に紛れてしまいます。

原理的には指定した距離の値以降になるとマテリアルにフォグとして設定した色を加算しているだけです。

### 44.2 設定値

設定名	機能
fogColor	霧の色
fogStartDistance	霧の効果を適応し始める距離。 ここから効果が終点まで徐々に適応される。
fogEndDistance	霧の効果の終点距離。これ以降は fogColor で設定した色に オブジェクトが塗りつぶされる。
fogDensityExponent	霧の効果の強さ。値を上げると明るくなるが fogEndDistance 以降は fogColor の色が適応され効果はない

### 44.3 設定例

黒でフォグを設定したもの。

リスト 43.1:

```
let scene = SCNScene()  
scene.fogColor = UIColor.black  
scene.fogStartDistance = 0
```

```
scene.fogEndDistance = 4  
scene.fogDensityExponent = 1
```

背景を白にすると遠くのジオメトリを黒く塗っているだけの処理というのがわかります。



図 44.1 背景を黒くしたフォグの効果

# 第45章

## シーンの Point of View

---

シーンの機能では、Point of View という機能があり、シーン内でのカメラを設定、他のカメラへ表示を変更することができます。

SCNView の allowsCameraControl = true にするとフリックでカメラの視点を変更できるフリーカメラとなります。内部ではシーンで設定しているカメラがフリックされた瞬間に Point of View を使用してフリーカメラ変更されています。

また、シーン内で複数のカメラを使用したい場合にも Point of View 便利です。

### 45.1 コードでの設定方法

SCNView の pointOfView にカメラのノードを設定するだけです。シーンにカメラがある場合は、View のシーン設定 (scnView.scene = scene) 後に pointOfView を設定を行わないとシーン側のカメラが優先されてしまいます。

リスト 44.1: Point of View の設定

```
scnView.pointOfView = cameraNode  
}
```

また、カメラ切り替えをアニメーションしたい場合は、pointOfView がアニメーション可能ですので、SCNTransaction や Core Animation で設定するだけです。

動作がおかしくなる可能性があるため、アニメーションさせる場合は 2 つのカメラをシーン上に addChild で追加してください。

```
//list[Point of View Anim][Point of View でのアニメーション]{  
scene.rootNode.addChildNode(cameraNode)  
scene.rootNode.addChildNode(cameraNode2)  
  
scnView.pointOfView = cameraNode  
  
// アニメーション
```

```
SCNTransaction.begin()
SCNTransaction.animationDuration = 4.0

// アニメーションさせたいカメラを設定する
scnView.pointOfView = cameraNode2

SCNTransaction.commit()
```

## 45.2 カメラの適応について

シーン内のカメラは最初に設定されたカメラがシーディングラフ上では上位にあたり、メインのカメラとして設定されます。

View のシーン設定が行われるまではシーン内で表示するカメラの設定は可能ですが、シーン設定がされてしまうと使用しているカメラをシーンから消してもシーディングラフ上で次に上位にあるカメラへは変更されません。

View の pointOfView はシーン設定前に設定すると、シーン設定時のカメラ情報で上書きされてしまいますが、シーン設定後はこちらで設定することで、指定したカメラに表示を変更することができます。

# 第46章

## CameraController

---

CameraController は iOS 11 などの新しい OS で増えた新機能です。

SCNView の allowsCameraControl を true にした際、フリックやピンチなどでカメラ操作ができますが、iOS 11 ではこちら振る舞いを開発者側から設定することが可能になりました。SCNView の defaultCameraController の設定を変更し振る舞いを替えます。

### 46.1 インタラクションモード

CameraController でどのような操作ができるかの紹介。

画面フリックの操作は用意されているインタラクションモード変更可能で、コードから変更する場合、Xcode の Scene Editor にはないコントロールもあります。

- fly
- orbitTurntable
- orbitAngleMapping
- orbitCenteredArcball
- orbitArcball
- pan
- truck

#### 46.1.1 fly

Xcode の Scene Editor で使用した場合、W、A、S、D のキーでゲームのように、X、Y 軸移動することができます。iOS ではキー入力が動作しません。

### 46.1.2 orbitTurntable

これまでのもので、左右フリックで画面に対して Y 軸（スクリーン座標の Y 軸）で回転、上下で画面に対して X 軸回転します。

### 46.1.3 orbitAngleMapping

左右フリックで画面に対して Y 軸回転、上下で画面に対して X 軸回転します。

### 46.1.4 orbitCenteredArcball

ヘッダーファイルにコメントがないため、詳しい動作はわかりません。

現状 orbitArcball と同じ振る舞いをします。

### 46.1.5 orbitArcball

orbitAngleMapping と同じですが、上下フリックの振る舞いが逆です。また、早くフリックすると戻る力が働きます。

### 46.1.6 pan

左右フリックで画面に対して X 軸移動、上下で画面に対して Y 軸移動します。

### 46.1.7 truck

左右フリックで画面に対して X 軸移動、上下で画面に対して Z 軸移動（ズームイン、アウト）します。

## 46.2 CameraController 設定値

- interactionMode
- inertiaEnabled
- inertiaFriction

- `isInertiaRunning`
- `maximumHorizontalAngle`
- `maximumVerticalAngle`
- `minimumHorizontalAngle`
- `minimumVerticalAngle`
- `pointOfView`
- `target`
- `automaticTarget`
- `worldUp`
- `delegate`

### 46.2.1 `interactionMode`

先に紹介しているインタラクションモードのどれを使用するか設定します。

### 46.2.2 `inertiaEnabled`

カメラをフリックで動かした後に慣性が働くか設定します。`false` にした場合指を離すとすぐ止まります。

### 46.2.3 `inertiaFriction`

カメラをフリック後の動きに空気抵抗がどのくらい働くか決めます。初期値は 0.05 で、動きがなだらかに止まり、0.0 にすると抵抗がなくなるため回り続けます。

### 46.2.4 `isInertiaRunning`

カメラをフリックで動かした後に慣性が働いているか、止まっているかを `Bool` で返します。フリックしている時ではないの注意してください。

### 46.2.5 **minimumVerticalAngle**、**maximumVerticalAngle** **minimumHorizontalAngle**、**maximumHorizontalAngle**

Fly もしくは Turntable モードにした場合、カメラを回転する際、緯度方向 (VerticalAngle) と経度方向 (HorizontalAngle) の角度の制限設定をします。

minimum と maximum が 0 の場合は効果が現れず、maximum より minimum の値の方を小さくするように設定しないと動作しません。

フリックをしたワールド座標で、VerticalAngle の範囲は -90 ~ 90、HorizontalAngle は -180 ~ 180 です。設定する値はラジアンではないので注意してください。

### 46.2.6 **pointOfView**

シーンの pointOfView と同様です。

### 46.2.7 **target**

orbit 回転するタイプのものの回転軸の位置を指定します。

### 46.2.8 **automaticTarget**

beginInteraction が呼ばれた際に target の位置を自動で更新するか否かの設定。

### 46.2.9 **worldUp**

Fly もしくは Turntable モードにした場合、回転の方向の指定をする。初期値は Y 軸回転で SCNVector3(0, 1, 0)。

### 46.2.10 **delegate**

デリゲートを設定し、以下の関数を実行できるようにします。

### **func cameraInertiaWillStart(for cameraController: SCNCameraController)**

フリックしカメラ回転後に慣性が働いた時にこの関数が呼ばれます

### **func cameraInertiaDidEnd(for cameraController: SCNCameraController)**

フリックしカメラ回転後に慣性が働き止まった時にこの関数が呼ばれます

## 46.3 CameraController のメソッド

動作を変更するメソッドとインタラクションが起こった際に実行されるメソットがあります。

### 46.3.1 **clearRoll()**

回転を最初の状態に戻します。

### 46.3.2 **stopInertia()**

慣性が動きを止めます。

### 46.3.3 **translateInCameraSpaceBy(x: Float, y: Float, z: Float)**

カメラを移動させます。移動する座標はカメラのローカル座標です。

### 46.3.4 **frameNodes([SCNNode])**

指定したノードがカメラの表示エリアに入るよう、カメラのズームを変更します。

### 46.3.5 **rotateBy(x: Float, y: Float)**

カメラを回転させます。

### 46.3.6 dolly(by: Float, onScreenPoint: CGPoint, viewport: CGSize)

画面上のポイントを元にカメラを移動してズームします。

### 46.3.7 dolly(toTarget: Float)

ターゲットへカメラが移動してズームします。

### 46.3.8 roll(by: Float, aroundScreenPoint: CGPoint, viewport: CGSize)

画面上のポイントを中心にカメラを回転させます。

### 46.3.9 rollAroundTarget(Float)

ターゲットの軸の周りで回転します。

### 46.3.10 インタラクションが起こった際に動作させる関数

- beginInteraction(CGPoint, withViewport: CGSize)
- continueInteraction(CGPoint, withViewport: CGSize, sensitivity: CGFloat)
- endInteraction(CGPoint, withViewport: CGSize, velocity: CGPoint)

画面フリック等のインタラクション開始時、動作中、終了後で設定する。

## 46.4 SCNCameraControlConfiguration

SCNCameraController の動作を設定するパラメーターがあります。

- allowsTranslation
- autoSwitchToFreeCamera
- flyModeVelocity
- panSensitivity

- rotationSensitivity
- truckSensitivity

### 46.4.1 allowsTranslation

false の場合、2 本指でカメラを移動するとターゲットの起点が指の位置になります。

### 46.4.2 autoSwitchToFreeCamera

指を動かし CameraControl を行うと設定しているカメラからフリーカメラに変更されます。通常は自動的に変更されるのですが autoSwitchToFreeCamera を false で設定すると変更されなくなります。

例えば、画面をダブルタップするとフリーカメラから設定しているカメラに戻るのですが、false すると機能しなくなります。

### 46.4.3 flyModeVelocity

fly 移動距離を m/s 単位で変更します。

### 46.4.4 panSensitivity、rotationSensitivity、truckSensitivity

pan、rotation、truck で動作する感度を上げます。初期値は 0.1 です。例えば、orbitTurntable モードで rotationSensitivity を 10 にすると、画面フリックするとカメラが高速で回転します。

## 46.5 ARKit での CameraController の振る舞いについて

ARKit ではデバイスのセンサーとカメラを使用して仮想空間のカメラを作成しているため、allowsCameraControl を使用すると表示がおかしくなる可能性があります。

# 第47章

## Screen Space Ambient Occlusion (SSAO)

---

iOS 11 から実装された機能で、アンビエントライトの光が塞がれ届かない隅やポリゴン面の交差、近接されている所に、やわらかな陰をリアルタイムで描画しリアル表現をします。

溝に対して陰を与えるため光が当たる部分にも影つき、光学的にはありえない現象ですので擬似的なものだと思ってください。

### 47.1 Screen Space Ambient Occlusion の仕組み

画面内でカメラから物体までの距離である深度情報をオフラインレンダリングで画像作成し、そのテクスチャを元にジオメトリの遮蔽される状態を計算します。最後に描画するジオメトリの情報と陰となる画像を合成することで、通常、ライトとジオメトリから計算を行うアンビエントオクルージョンより高速な処理が可能になります。

### 47.2 SSAO の設定方法

設定する際の注意点があり、以下を作業を行わないと設定が反映されません。

- ・アンビエントライトが 1 以上必要。アンビエントライトの intensity は 0 可能
- ・適応するマテリアルを Physically Based で設定（要 Metal）
- ・カメラの screenSpaceAmbientOcclusionIntensity を 0 以上に設定

### 47.3 注意点

カメラ単位での設定になりますので、他のカメラを使用する場合は同じ設定をする必要があります。例えば Xcode の Scene Editor では配置したカメラでのみ設定変更可能であるため、Perspective や Top などのカメラではその効果が現れません。

また、SSAO はカメラの深度情報から陰の作成を行います。そのためカメラから隠れたジオメトリへの影響や、隠れたジオメトリが表示されている他のジオメトリへ陰の影響を与えることはできません。

### 47.4 設定値

#### 47.4.1 Intensity

アンビエントオクルージョンの強度を設定します。

初期値が 0 のため、0 以外に設定しないと効果が出ません。

#### 47.4.2 Radius

シーン単位でのアンビエントオクルージョン半径を設定します。

値を大きくすると陰の範囲が増えます。

#### 47.4.3 Bias

シーン単位での陰のバイアスを設定します。

値を増やすとアンビエントオクルージョンがない状態に近くなります。

#### 47.4.4 Depth Threshold

シーン単位での奥行きから陰のぼかしの閾値を設定します。

0 すると陰が荒くなり、増やすとぼかしが大きくなります。

陰の表示が綺麗ではない場合、値を増やすと改善する可能性があります。

### 47.4.5 Normal Threshold

陰に対してぼかしの閾値を決定します。

DepthThreshold の後に計算されているようなので、Depth Threshold で調整し、こちらを調整した方がよさそうです。

## 47.5 コードでの記述

リスト 46.1: SSAO:初期値のままで Intensity を 2 にしたもの

```
let camera = SCNCamera()
camera.screenSpaceAmbientOcclusionIntensity = 2
camera.screenSpaceAmbientOcclusionRadius = 5
camera.screenSpaceAmbientOcclusionBias = 0.03
camera.screenSpaceAmbientOcclusionDepthThreshold = 0.2
camera.screenSpaceAmbientOcclusionNormalThreshold = 0.3
```

# 第48章

## 効果音やBGMの再生

---

Core AudioなどiOS SDKの機能で音や音楽が使用できますが、SceneKitのオーディオ再生機能を使用することもできます。

### 48.1 音を再生する種類

SceneKitでは音を再生する方法が2つあり、CNAudioSourceとSCNAudioPlayer。SCNAudioSourceでオーディオファイルの設定し、SCNAudioPlayerではSCNAudioSourceやAVAudioNodeを設定します。

- ・ノードのaddAudioPlayerにSCNAudioSource(AVAudioNode)を設定したSCNAudioPlayerで再生する
- ・SCNActionのplayAudioでSCNAudioSourceを指定して再生する

SCNAudioPlayerは主に音楽を再生し続けるBGMなど、SCNActionのplayAudioは効果音などに使用します。

### 48.2 SCNAudioSource

#### **init?(named: String)**

オーディオファイル名の指定します。

#### **init?(fileNamed: String)**

main bundleからのオーディオファイル名の指定します。

#### **init?(url: URL)**

URLからのオーディオファイル名の指定します。

### isPositional

初期状態では、オブジェクトの距離によってオーディオの音量やリバーブを自動的に変化されます。false にすると場所に関係なく一定で再生されます。

### load()

事前読み込み用メソッド。shouldStream プロパティの値が true の場合は無効になります。

### volume

0 から 1 の間で音量を設定します。

### rate

再生スピード。値を大きくすると再生速度が速くなります。

### reverbBlend

リバーブ (お風呂場のような音響効果) を付加します。

### loops

繰り返し再生するか否か。初期値は false で 1 回のみの再生します。

### shouldStream

true の場合はソースファイルから直接読み込み、オーディオバッファデータにロードを行いません。

## 48.3 SCNAudioPlayer

SCNAudioSource か AVAudioNode のいずれかを設定します。

### init(source: SCNAudioSource)

SCNAudioSource を設定し初期化します。

### **init(avAudioNode: AVAudioNode)**

AVAudioNode を設定し初期化します。

### **audioSource**

SCNAudioSource を設定します。

### **audioNode**

AVAudioNode を設定します。

### **willStartPlayback**

SCNAudioPlayer 再生前に設定したブロックが呼ばれます。ループの場合は最初に戻った際、毎回呼ばれるので注意が必要です。

### **didFinishPlayback**

再生が止まった時に設定したブロックが呼ばれます。

## 48.4 SCNNode で SCNAudioPlayer を扱うための機能

### **addAudioPlayer(SCNAudioPlayer)**

SCNAudioPlayer を追加します。複数可能。

### **audioPlayers**

SCNAudioPlayer が配列で返ります。読み取り専用。

### **removeAudioPlayer(SCNAudioPlayer)**

指定した設定されている SCNAudioPlayer を消し再生を停止する

### **removeAllAudioPlayers()**

全ての設定されている SCNAudioPlayer を消し再生を停止する

## 48.5 SCNAudioPlayer 使用をして再生する

SCNNode で SCNAudioPlayer を設定し、設定後、即再生されます。途中で止めたい場合は、removeAllAudioPlayers か、removeAudioPlayer で設定した SCNAudioPlayer を選択し、SCNAudioPlayer を消します。

リスト 47.1: SCNAudioPlayer 使用をして再生する

```
let audioNode = SCNNode()
audioNode.name = "audioNode"
audioNode.position = SCNVector3(0,0,0)
scene.rootNode.addChildNode(audioNode)

let audioSource = SCNAudioSource(named: "bgm.mp3")
audioSource?.loops = true

let audioPlayer = SCNAudioPlayer(source: audioSource!)
audioNode.addAudioPlayer(audioPlayer)

audioPlayer.willStartPlayback = {
    print("willStartPlayback")
}

audioPlayer.didFinishPlayback = {
    print("didFinishPlayback")
}
```

## 48.6 SCNAction で SCNAudioSource を設定する

SCNAction の playAudio を使用して SCNAudioSource を適応して再生します。

waitForCompletion が true の場合は再生持続時間が SCNAction の長さになり、false の場合は再生後すぐ完了され、他のアクションがあれば再生中でも移行されます。

キャラクターの攻撃など連續で音を出したい場合は false を指定します。

リスト 47.2: SCNAction で SCNAudioSource を設定する

```
let audioNode = SCNNode()
audioNode.name = "audioNode"
audioNode.position = SCNVector3(0,0,0)
scene.rootNode.addChildNode(audioNode)

let audioSource = SCNAudioSource(named: "sound.mp3")

audioNode.runAction(SCNAction.playAudio(audioSource!, waitForCompletion: true))
```

### 48.7 注意点

SCNAudioPlayer や SCNAction で SCNAudioSource を使用してシーン内に音を最初に再生する場合、負荷の問題なのか何かは不明ですが、音源再生時に画面全体で処理落ちが起こります。

理由はわかりませんが、いずれか音源が鳴ると他の load() が効くようです。

シーン表示直後に効果音を鳴らしたい場合は、その前にBGMを流すか、無音の音源を流すなど何か音を鳴らす必要があるようです。

# 第49章

## テッセレーションとサブディビジョンサーフェイス (A9 以降の端末のみ)

---

ポリゴンの構成（トポロジー）を細かく、もしくは荒く再構成して、ジオメトリの見た目を変更する機能が 2 つあります。

3 世代目の GPU (A9,A9X) から使用できる機能であるため、古い端末では動作しません。

### 49.1 テッセレーション

テッセレーションはポリゴンを細かく分割する技術で、iOS 10 の Metal で導入され、iOS 11 から SceneKit でも使用できるようになりました。彫刻のように 3DCG のモデリングをするデータの Displacement マップを適応するために使用したり、カメラの距離によってポリゴンを細かくしたり、荒くしたりする場合に使用します。

設定的には SCNGeometry の tessellator へ SCNGeometryTessellator を設定することで動作します。以下に、設定値の説明を書きます内部計算が複雑なため設定がわかりづらいです。tessellationFactorScale と isAdaptive、isScreenSpace を設定し、他は初期値でも動くためそこから試してみるとよいかもしれません。

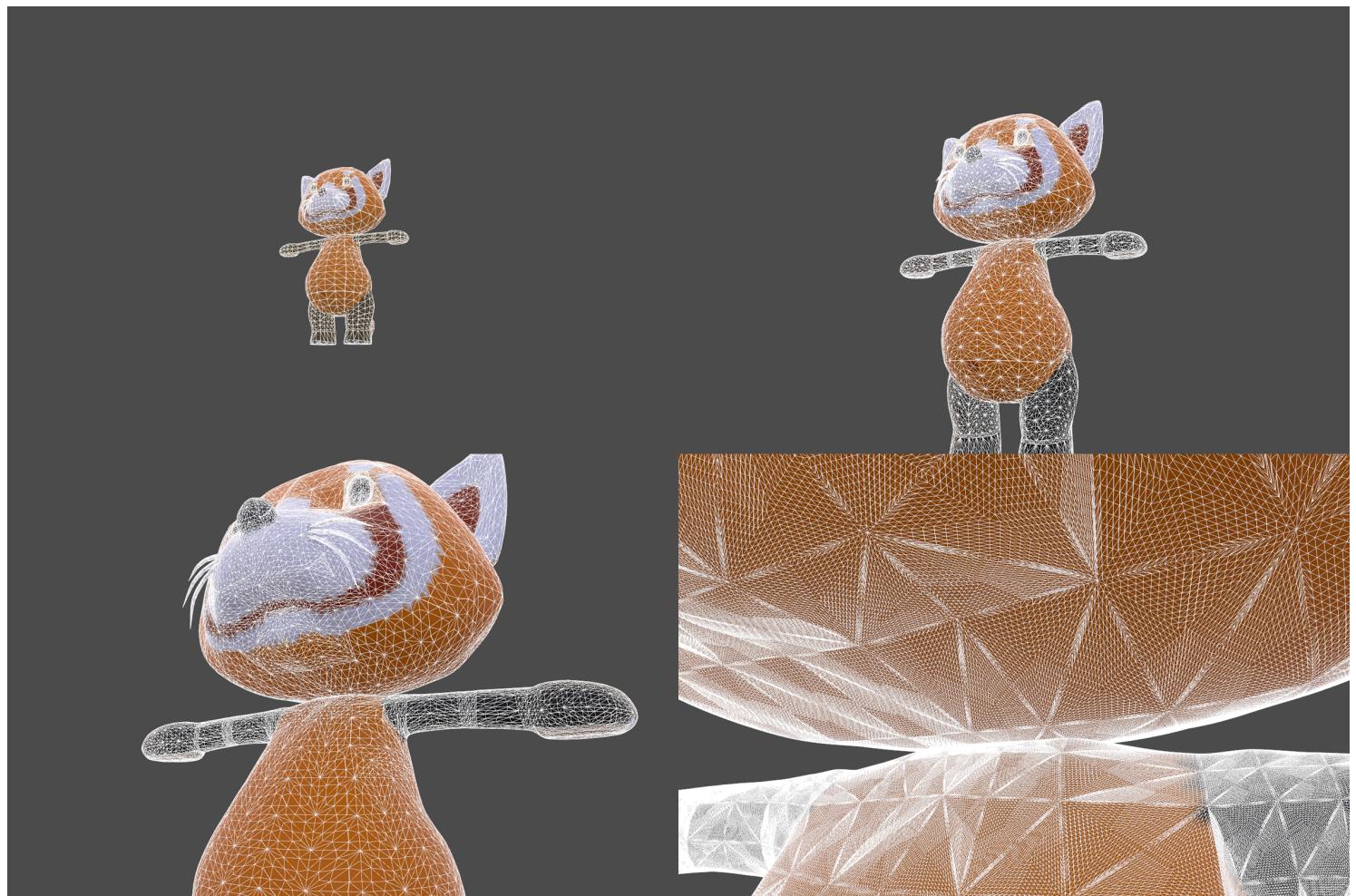


図49.1 設定値 `isScreenSpace` を `true` にしてカメラの距離でテッセレーションを適応

## 49.2 テッセレーションの設定値

- `tessellationFactorScale`
- `tessellationPartitionMode`
- `edgeTessellationFactor`
- `insideTessellationFactor`
- `isAdaptive`
- `isScreenSpace`
- `maximumEdgeLength`
- `smoothingMode`

## tessellationFactorScale

ポリゴンを分割する際、大きさを設定します。初期値は1で、値を大きくするとより細かく分割されます。

## tessellationPartitionMode

テッセレーションのパーティションモードを指定します。テッセレーションはエッジから細分化しますが、そこで使用されるセグメントの数と間隔を計算するための方法をどれに設定するかを決めます。

- pow2 (MTLTessellationPartitionMode.pow2)
- integer (MTLTessellationPartitionMode.integer)
- fractionalOdd (MTLTessellationPartitionMode.fractionalOdd)
- fractionalEven (MTLTessellationPartitionMode.fractionalEven)

pow2はテッセレーションレベルが最も近い整数nに切り上げられ、nは2の累乗です。UV空間内の等しい長さのn個のセグメントに分割されます。

integerはpow2の累乗しないもので、こちらが初期値になります。

fractionalOddはテッセレーションレベルを最も近い奇数整数nに切り上げられます。nが1の場合はエッジが細分化されず、それ以外の場合対応するエッジは長さの等しいn-2個のセグメントと、他のセグメントより短い同じ長さで2つセグメントに分割されます。もっと細かい条件があるので複雑であるため割愛させていただきます。

fractionalEvenテッセレーションレベルは最も近い偶数の整数nに切り上げられます。

## edgeTessellationFactor

ジオメトリのエッジに対してテッセレーション係数（分割数）を設定します。isAdaptiveを設定している場合はこの効果が無視されます。

## insideTessellationFactor

内部のテッセレーション係数を設定します。isAdaptive を設定している場合はこの効果が無視されます。

## isAdaptive

テッセレーションが均一に行われるか、形状に合わせて最適化されるかを決めます。初期値は false で均一に行われます。

## isScreenSpace

isAdaptive が true の時に効果が現れ、テッセレーションの分割をカメラの位置で変更を行うかを設定します。初期値は false で均一に行われます。

## maximumEdgeLength

isAdaptive が true である場合に使用することができ、分割するエッジの最大の長さを設定します。初期値は 1 でメートル単位となりますが、isScreenSpace が true である場合はピクセル単位になります。

## smoothingMode

設定は以下の 3 つで、初期値は none です。

- none
- pnTriangles
- phong

pnTriangles は頂点の座標と法線から分割し曲面を生成し、phong は生成されている 3 つの頂点の接する平面を投影し、投影の座標を使用してポリゴンの分割します。

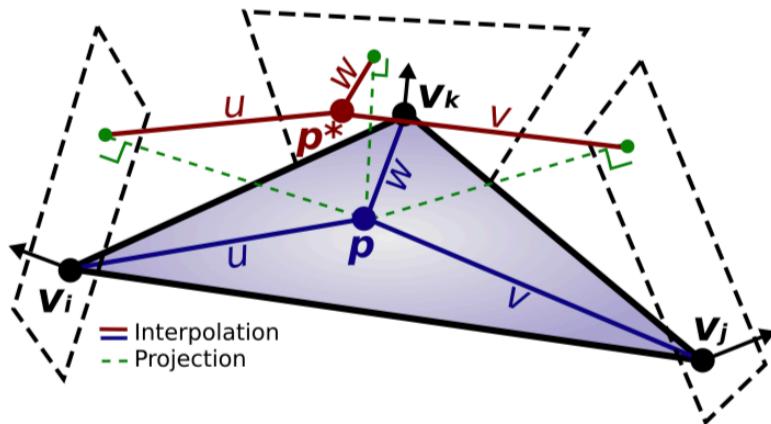


図49.2 phong のアルゴリズム

画像は「Phong Tessellation」 Tamy Boubekeur Marc Alexa TU Berlin, Page 2 より参考。<sup>\*1</sup>

### 49.3 テッセレーションの設定方法

全て値を設定していませんが、サブディビジョンサーフェイス設定することで、カメラを近づけたり、遠ざけたりすると、アダプティブにテッセレーションが行われポリゴンのトポロジーが動的に変化します。

リスト48.1: テッセレーションの設定

```
let cube = scene.rootNode.childNode(withName: "Cube", recursively: true)!

let tessellator = SCNGeometryTessellator()
tessellator.tessellationFactorScale = 14
tessellator.isAdaptive = true
tessellator.isScreenSpace = true
tessellator.smoothingMode = .phong

cube.geometry?.subdivisionLevel = 1

cube.geometry?.tessellator = tessellator
```

<sup>\*1</sup> <http://perso.telecom-paristech.fr/%7Eboubek/papers/PhongTessellation/>

## 49.4 サブディビジョンサーフェイス

ポリゴンメッシュを規則的に分割することで曲面を作成する技術です。Catmull - Clark のオープンソース版 OpenSubDiv という手法で実装されています。

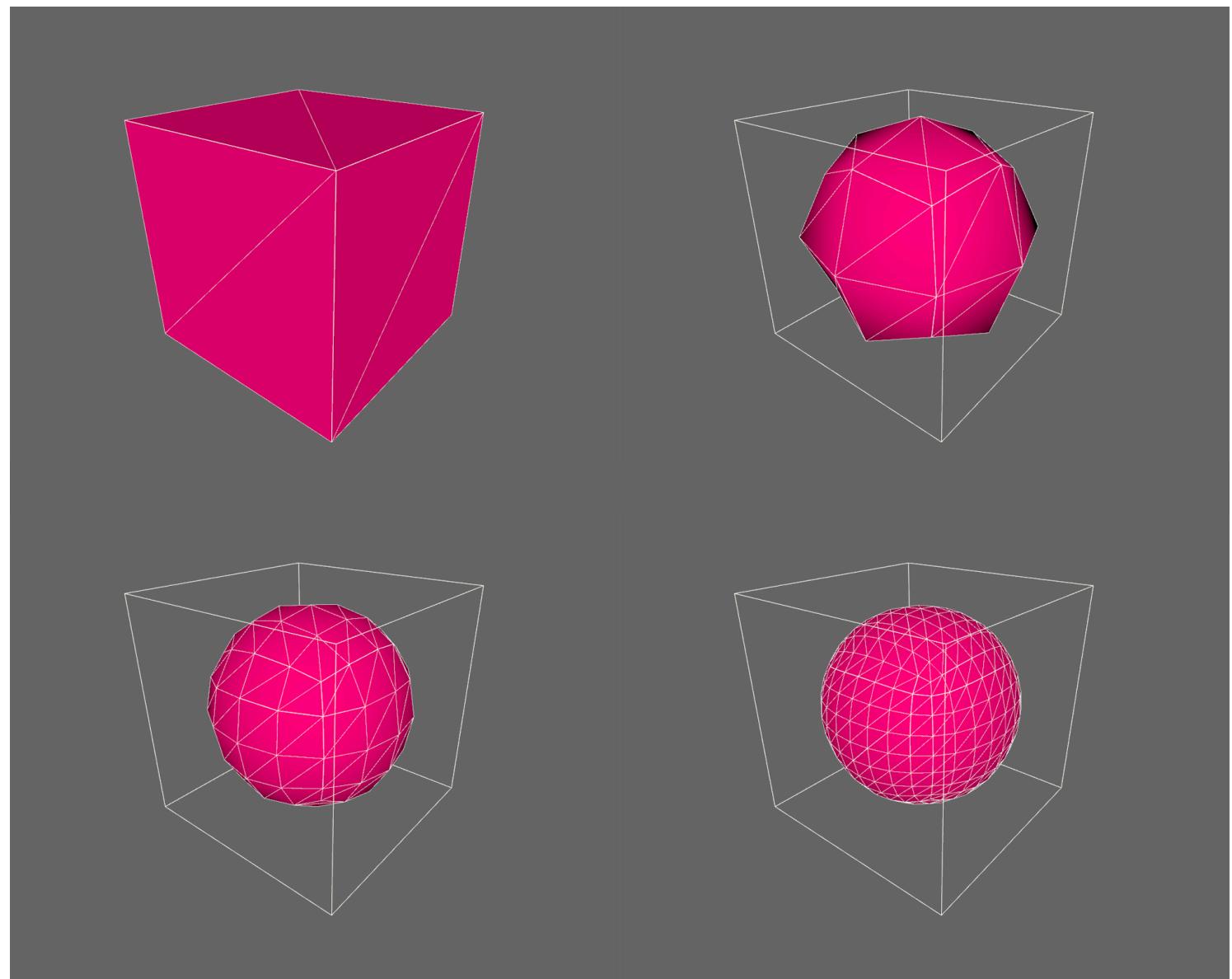


図 49.3 Sub Division Level : 0, 1, 2, 3

## 49.5 サブディビジョンサーフェイスの設定方法

設定値は SCNGeometry の subdivisionLevel で分割数を設定するだけです。テッセレーションが有効になっている場合、カメラの位置に最適化された曲面が効率よく生成できます。

リスト 48.2: サブディビジョンサーフェイスの設定

```
cube.geometry?.subdivisionLevel = 4
```

wantsAdaptiveSubdivision が false になっている場合、テッセレーションは適応されません。

## 49.6 頂点やエッジに対してクリーズで適応度合いの変更する

ポリゴンの頂点やエッジに対してクリーズ (crease) が設定可能で、3DCG DCC ツールのように、頂点やエッジでの適応度合いを変更することができるようです。試したこと正しい設定がわからず効果が現れませんでした。

## 49.7 サブディビジョンサーフェイス設定での注意点

サブディビジョンサーフェイスの特性上、三角ポリゴンの部分は綺麗な曲面になりづらいです。そのため、SCNBox など三角ポリゴンで構成されているビルトインジョメトリでは、意図しない形状になる可能性があります。

# 第50章

## ノードの最適化

ノードの最適化を解説します。

### 50.1 Clone (複製)

同じジオメトリを複数設定するために、シーンのノードを何度も生成するとドローコールが増えたり、その他で処理がかかってしまいます。ノードの命令で `clone` というものがあり、内容を複製して描画処理で実行するため負荷が軽くなり、本書では物理アニメーションで使用しています。

#### 50.1.1 Clone 使用例

リスト 49.1: Clone を使用

```
let ballNode = SCNNNode(geometry: SCNSphere(radius: 1))
scene.rootNode.addChildNode(ballNode)

// ここを SCNNNode(geometry: SCNSphere(radius: 1)) から
// ballNode.clone() にすることでドローコールが 1 減る
let ballNode2 = ballNode.clone()
ballNode2.position = SCNVector3(2.5, 0.0, 0.0)
scene.rootNode.addChildNode(ballNode2)
```

#### 50.1.2 Clone での注意点

`Clone` を使用すると、複製元のノードの情報を参照します。そのため、複製先のノードのパラメーターを変更すると、複製元も変更されてしまい、例えばマテリアルの `diffuse` の色を変える場合は、複製元も変わってしまいます。

中身を編集したい場合は、ドローコールが増えることになりますが、新規で作成するか、内容をコピーしてください

リスト 49.2: 内容をコピーする

```
// 1つ目の球
let ballNode = SCNNNode(geometry: SCNSphere(radius: 1))
ballNode.geometry?.firstMaterial?.diffuse.contents = UIColor.red
scene.rootNode.addChildNode(ballNode)

// 2つ目の球
let ballNode2 = SCNNNode()
ballNode2.position = SCNVector3(2.5, 0.0, 0.0)

// ジオメトリのコピー
ballNode2.geometry = ballNode.geometry?.copy() as? SCNGeometry

// マテリアルのコピー
let newMaterial = ballNode.geometry!.firstMaterial?.copy() as! SCNMaterial

ballNode2.geometry?.firstMaterial = newMaterial
ballNode2.geometry?.firstMaterial?.diffuse.contents = UIColor.blue

scene.rootNode.addChildNode(ballNode2)
```

## 50.2 階層化しているノードを一つにまとめる

ノードに対してチルドノードを追加し、さらにチルドノードを何回か追加し続けて入れ子にしていくと、シンググラフ上でノードを探索して処理するが増えるため、全体のパフォーマンスが悪くなります。

SCNNode に用意されている `flattenedClone` を使用すると、入れ子になっているノードを 1 つにすることができます。

### 50.2.1 注意点

- 一気に 1 つのノードにするため内容がおかしくなる可能性があります。中身がおかしくなってないか確認しましょう。

- カメラやライトなどは削除されます。シーン内で使用している場合は気をつけてください。

### 50.2.2 使用例

リスト 49.3: flattenedClone でノードを一つにまとめる

```
let ballNode = SCNNNode(geometry: SCNSphere(radius: 1))
ballNode.geometry?.firstMaterial?.diffuse.contents = UIColor.red

let n1 = SCNNNode()
n1.addChildNode(ballNode)

let n2 = SCNNNode()
n2.addChildNode(n1)

let n3 = SCNNNode()
n3.addChildNode(n2)

// 1つのノードになって addChildNode される
scene.rootNode.addChildNode(n3.flattenedClone())
```

## 50.3 他のシーンファイルを参照として配置する

他のシーンファイルを参照として配置することができますが、マテリアルなど一部パラメーターの変更ができません。使用用途はあまりわかりませんが、こちらは Scene Editor 用に作成されたものだと思われます。

初期設定では、設定後 load() を使用する必要があり、unload() を使用するとシーンから削除できます。

リスト 49.4: SCNReferenceNode で参照

```
guard let path = Bundle.main.path(forResource: "scene", ofType: "scn") else
{ return }

let referenceNode = SCNReferenceNode(url: URL(fileURLWithPath: path))!
```

```
referenceNode.load()  
  
scene.rootNode.addChildNode(referenceNode)
```

# 第51章

## iOS 11 で追加されたノードの機能

---

位置、回転、拡大縮小などの操作。GameplayKit の GKEntity の直接設定、tvOS でノードのフォーカスする設定（反応なし、遮断する、フォーカスする）を決めるなどが追加されました。

位置、回転、座標関連などの操作でいくつか簡単に操作できるものが用意され、SCNVector3 だけではなく SIMD の命令も追加されています。

### 51.1 SIMD

1 つの命令を同時に複数のデータに適用し、並列処理で高速に計算を行う SIMD で設定できるようになりました。以下が SceneKit に対応しているものです。

- SCNVector3>simd\_float3
- SCNVector4>simd\_float4
- SCNVector4>simd\_quatf (simd\_float4)
- SCNMatrix4>simd\_float4x4 (simd\_float4, simd\_float4, simd\_float4, simd\_float4)

simd\_float3 は `simd::float3` として C++ や Metal で使用でき、simd\_float4 などは Metal で使用できません。

また、SIMD の命令から SceneKit とやり取りする際に時間がかかる場合があるかもしれません。状況によっては、SCNVector3、4 や SCNMatrix4 に変更してみてください。

## 51.2 プロパティ

### 51.2.1 既存のものに SIMD が使えるようになったもの

eulerAngles、orientation、pivot、position、rotation、scale、transform に対応しています。simdEulerAngles は eulerAngles 同様に、ロール (X 軸)、ヨー (Y 軸)、ピッチ (Z 軸) で適応され、worldTransform はワールド座標の読み取りのみですが、simdWorldTransform はワールド座標の設定ができます。

- simdEulerAngles: simd\_float3
- simdOrientation: simd\_quatf
- simdPivot: simd\_float4x4
- simdPosition: simd\_float3
- simdRotation: simd\_float4
- simdScale: simd\_float3
- simdTransform: simd\_float4x4
- simdWorldTransform: simd\_float4x4

### 51.2.2 SceneKit のプロパティで追加されたもの

ノードのワールド座標が情報が簡単に取得できるようになりました。

#### worldFront: SCNVector3

読み取り用で、SCNVector3(x: 0.0, y: 0.0, z: -1.0) を返します。

#### worldRight: SCNVector3

読み取り用で、SCNVector3(x: 1.0, y: 0.0, z: 0.0) を返します。

#### worldUp: SCNVector3

読み取り用で、SCNVector3(x: 0.0, y: 1.0, z: 0.0) を返します。

### worldPosition: SCNVector3

ワールド座標で位置を SCNVector3 で取得、または設定することができます。

以下の例では、X 軸 2 にありますが、node1 の worldPosition.x で 0 に指定して移動しています。ワールド座標を取得してローカル座標を引かなくとも済むようになりました。

リスト 50.1: worldPosition

```
let node1 = SCNNNode()  
node1.position = SCNVector3(1,1,1)  
  
node1.worldPosition.x = 0
```

### worldOrientation: SCNQuaternion

worldPosition の回転版です。

ワールド座標で回転をクオータニオンでの SCNVector4 で取得、または設定することができます。

#### SIMD 版

- simdWorldFront: simd\_float3
- simdWorldOrientation: simd\_quatf
- simdWorldPosition: simd\_float3
- simdWorldRight: simd\_float3
- simdWorldUp: simd\_float3

## 51.3 タイププロパティ

worldFront などと同じ振る舞いです。

- localFront: SCNVector3
- localRight: SCNVector3
- localUp: SCNVector3

- simdLocalFront: simd\_float3
- simdLocalRight: simd\_float3
- simdLocalUp: simd\_float3

## 51.4 メソッド

### 51.4.1 **convertVector(SCNVector3, to: SCNNNode?), convertVector(SCNVector3, from: SCNNNode?)**

SCNVector3 の座標系から指定されたノードの座標系に位置を変換し、ローカル座標を返します。to はノードの変換元となり、from はノードの変換先で、ノードが nil の場合はワールド座標から変換されます。

Apple が公開している ARKit のカメレオンのサンプルでは舌をカメラ手前に貼り付けるために使用します。

AR の関係上カメラが動くため、カメレオンの口の位置とカメラの座標から舌をはりつける位置を取得しています。

### 51.4.2 **localRotate(by: SCNQuaternion)**

指定したノードのローカル座標の回転のクオータニオンの SCNVector4 で取得し、自身の orientation へ取得したベクトルとともに適応されます。

自身が 45 度回転して、対象が 45 度回転している場合、90 度回転が行われます。

### 51.4.3 **localTranslate(by: SCNVector3)**

指定したノードのローカル座標の位置の SCNVector3 で取得し、自身の position へ取得したベクトルとともに適応されます。

自身が SCNVector3(-1, -1, 0) において、対象が SCNVector3(2,2,0) がある場合、local-Translate を使用すると SCNVector3(1, 1, 0) に移動します。

#### 51.4.4 **look(at: SCNVector3)、look(at: SCNVector3, up: SCNVector3, localFront: SCNVector3)**

SCNLookAtConstraint のように指定したノードの方向を向きます。at で指定した座標を向くように回転し、up は基準となる Y 軸方向を設定し、localFront は正面がどこかを設定します。

look(at: SCNVector3) は簡易版で up は worldUp (SCNVector3(0, 1, 0))、localFront (SCNVector3(0, 0, -1))。localFront が -1 なので、初期位置は後ろ向きになります。

#### 51.4.5 **rotate(by: SCNQuaternion, aroundTarget: SCNVector3)**

回転させる中心軸方向を指定して回転を行います。

振る舞いは SCNAction.rotate(by:,around:,duration:) の duration の時間設定とアニメーションがないバージョンです。

#### 51.4.6 **setWorldTransform(SCNMatrix4)**

worldTransform でワールド座標は設定できませんでしたが、こちらの命令を設定できるようになりました。

### 51.5 メソッド SIMD 版

SIMD には convertPosition という命令が追加されています。

また、simdWorldTransform でワールド座標の変更が可能であるため、setWorldTransform に該当するものはありません。

- simdConvertPosition(simd\_float3, from: SCNNNode?)
- simdConvertPosition(simd\_float3, to: SCNNNode?)
- simdConvertTransform(simd\_float4x4, from: SCNNNode?)
- simdConvertTransform(simd\_float4x4, to: SCNNNode?)
- simdConvertVector(simd\_float3, from: SCNNNode?)
- simdConvertVector(simd\_float3, to: SCNNNode?)

- simdLocalRotate(by: simd\_quatf)
- simdLocalTranslate(by: simd\_float3)
- simdLook(at: vector\_float3)
- simdLook(at: vector\_float3, up: vector\_float3, localFront: simd\_float3)
- simdRotate(by: simd\_quatf, aroundTarget: simd\_float3)

### 51.5.1 **simdConvertPosition(simd\_float3, from: SCNNode?), simdConvertPosition(simd\_float3, to: SCNNode?)**

直接ワールド座標を取得でき、simd\_float3 を返します。simd\_float3 は取得した座標からのオフセット値です。

以下の例では、X 軸 2, Y 軸 2 にある s ノードを配置し、ノード s2 は s のワールド座標から X + 1 したオフセットを取得して、そこに移動しています。

リスト 50.2: simdConvertPosition

```
let s = SCNNode(geometry:SCNSphere(radius: 0.2))
s.position = SCNVector3(2,2,0)
n.addChildNode(s)

let s2 = SCNNode(geometry:SCNSphere(radius: 0.2))
// SCNVector3(3,2,0) に移動される
s2simdPosition = s2simdConvertPosition(simd_float3(1, 0, 0), from: s)
scene.rootNode.addChildNode(s2)
```

to へ変更した場合は s から s2（デフォルトなのでシーンの原点）までの距離になるため、float3(-1.0, -2.0, 0.0) が返ります。X + 1 しているので float3(-2.0, -2.0, 0.0) から X - 1 引かれています。

# 第 52 章

## ジオメトリに Core Image のフィルターを使用する

---

ノードに対して、フィルターエフェクトを設定することが可能で、Core Image の命令数行で設定することができ非常に簡単に実装できます。

### 52.1 概要

Core Image のフィルター、Core Image Filter は Metal に統合されており、Metal のコマンドバッファへの設定が可能です。Metal に書き込まれているため、SceneKit のジオメトリにフィルター効果を簡単なコードの記述でき、そこそこ高速な処理で効果を適応することができます。

### 52.2 実行例

設定方法は CIFilter を作成して、ノードに filters の配列に適応するだけです。配列で指定するため、複数のフィルターを適応することができます。ノードに設定しますがジオメトリのアルファが考慮されるため、ジオメトリ部分のみフィルターが適応されます。（昔はジオメトリの大きさの矩形で設定されてた記憶がありますが記憶が曖昧です）

リスト 51.1: ノードに CIFilter を適応しモザイクにする

```
let box:SCNGeometry = SCNBox(width: 3, height: 3, length: 3, chamferRadius: 0.4)
let boxNode = SCNNNode(geometry: box)

let filter = CIFilter.init(name: "CIPixellate")
boxNode.filters = [filter!]

scene.rootNode.addChildNode(boxNode)
```

Apple のドキュメントで Core Image Filter の一覧があります。<sup>\*1</sup>

フィルター適応のために 2 つ画像が必要なものや、機械的に画像を作成するジェネレーターなどはありますが、ここにあるほとんどのものは使用できます。

---

<sup>\*1</sup> <https://developer.apple.com/library/content/documentation/GraphicsImaging/Reference/CoreImageFilterReference/index.html>

# 第53章

## 統計情報とデバッグオプション

### 53.1 統計情報

すでに使用していますが、SCNView の `showsStatistics` を `true` にすると View の下部に統計情報が表示されます。



図 53.1 統計情報

- 詳細表示用の「+」ボタン
- フレームレート（ゲージ）
- 使用しているライブラリ名 OpenGL(ES) or Metal
- フレームレート（文字）
- ドローコール（フレーム毎）
- 表示全体のポリゴン数

フレームレートは 60 fps 以上を最良としており、ゲージが緑色である事が望されます。(40 fps ぐらいが限度？)

ライブラリの選択はデフォルトだと自動になっていますが、一部機能は Metal 必須のものがあるため、GL ではなく Mt と表示されていなければ動かないので注意が必要です。

ドローコールは描画するものが画面に現れると 1 カウントされ、その隣は画面上に表示されているポリゴン数となっており、ドローコールが多い場合は描画に無駄が出ている可能性があります。また、統計情報の表示でドローコールが 1 使います。

### 53.1.1 詳細表示用の「+」ボタン

以下のような表示になり、1 フレームあたりの情報がリアルタイムで表示されます。

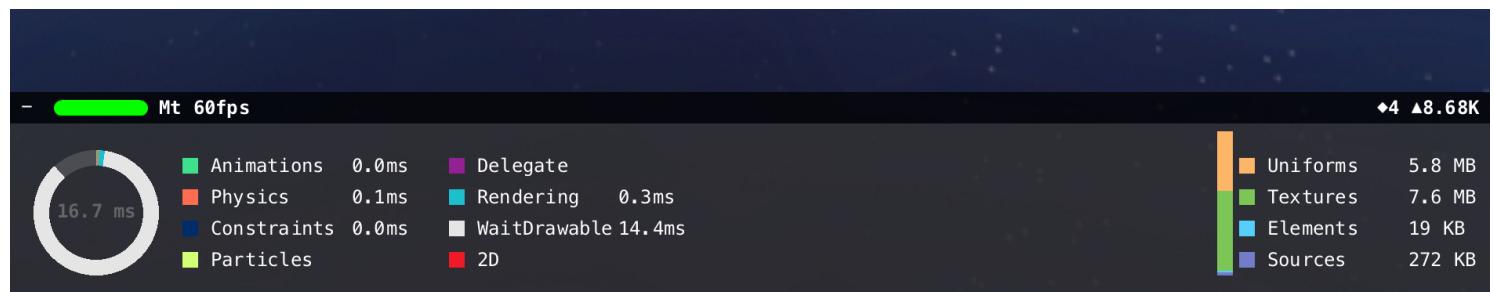


図 53.2 統計情報 詳細

60fps が理想となると 1 秒を 60 で割った 16.7ms が 1 フレームで処理する必要があり、アニメーション、物理アニメーション、コンストRAINT、パーティクル、各種デリゲート、レンダリング、2D（主に SpriteKit）を 16.7ms 以内に全て実行する必要があります

### 53.1.2 1 フレームでのレンダリングの流れ

SceneKit の処理は SCNSceneRenderer とそのデリゲートで行われ 1 フレームで以下のレンダリングループが順に動作が行われ、毎フレーム実行されます。

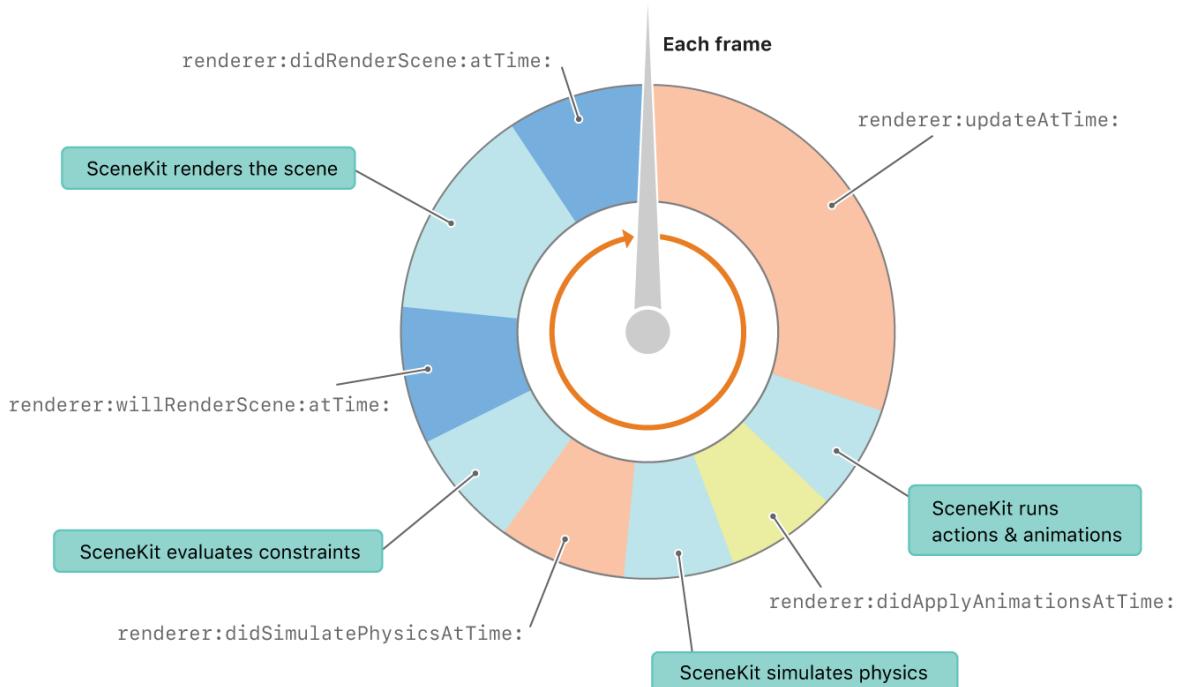


図 53.3 Rendering Loop

- View > renderer(\_: updateAtTime:) メソッドを呼び出す
- Scene > アクションやシーディングラフに関連付けられたアニメーションを実行
- View > renderer(\_: didApplyAnimationsAtTime:) メソッドを呼び出す
- Scene > 物理シミュレーションをシーン内の物理ボディに適用
- View > renderer(\_: didSimulatePhysicsAtTime:) メソッドを呼び出す
- Scene > コンストRAINTの処理を実行
- View > renderer(\_: willRenderScene: atTime:) メソッドを呼び出す
- Scene > 完了した処理を元にシーンをレンダリング
- View > renderer(\_: didRenderScene: atTime:) メソッドを呼び出す
- 最初に戻る

各種アニメーション > 物理アニメーション > コンストRAINT > レンダリング という流れで実行されます。ゲームなどでは、ゲームロジックの処理や通信の処理などを行うため 16.7ms 全てを描画に使用することができないので工夫が必要です。

上記で使用している SCNSceneRendererDelegate の renderer の命令を調整することでカスタムレンダリング処理をすることも可能です。

## 53.2 デバッグオプション

Xcode での Scene Editor など表示用に設定されているもので、SCNDebugOptions でデバッグオプションの設定されております。以下のものが設定可能です。

- showPhysicsShapes
- showBoundingBoxes
- showLightInfluences
- showLightExtents
- showPhysicsFields
- showPhysicsShapes
- showWireframe
- renderAsWireframe (iOS 11)
- showCameras (iOS 11)
- showConstraints (iOS 11)
- showCreases (iOS 11)
- showSkeletons (iOS 11)

### 53.2.1 showBoundingBoxes

ジオメトリなどのノードの周囲を四角で表示するバウンディングボックスを表示します。透明などジオメトリがわかりづらい時など、物体の位置が確認できます。

### 53.2.2 showLightInfluences

シーン内の各ライトの位置を表示します。

### 53.2.3 showLightExtents

シーン内の各ライトの影響を受ける箇所を表示します。

### 53.2.4 showPhysicsFields

シーン内の各物理フィールドの影響を受ける範囲を表示します。

### 53.2.5 showPhysicsShapes

PhysicsBody オブジェクトを持つノードの PhysicsShape（当たり判定）を表示します。 PhysicsBody がめり込んでしまったり、意図しない物理アニメーションが行われた時に使用すると便利です。

### 53.2.6 showWireframe

ワイヤーフレームレンダリングを使用して、シーン内のジオメトリを表示します。テッセレーション、サブディビジョンサーフェイスでポリゴンを分割率を確認する時などに使用します。

注意点として、通常でシェーディングするよりもワイヤーフレームレンダリングの方がレンダリングコストが掛かります。

### 53.2.7 renderAsWireframe (iOS 11)

showWireframe 近いのですがマテリアルが描画されなくなり、ワイヤーフレームにテクスチャを含めたマテリアルの色がつきます。

### 53.2.8 showCameras (iOS 11)

Xcode の Scene Editor 同様にカメラをワイヤーフレームで表示します。

### 53.2.9 showConstraints (iOS 11)

Slider コンストRAINTで制限されている場所よりノードが移動された場合、本来の移動すべき位置をワイヤーフレームで表示し、制限を受けて止まっている場所から実際の移動場所までバウンディングボックスで描画し表示します。

### 53.2.10 showCreases (iOS 11)

サブディビジョンサーフェイスのクリーズがちゃんと動作させられていないので動作不明ですが、試したところ、サブディビジョンサーフェイスがかかる前のジオメトリがワイヤーフレーム表示されるようです。

### 53.2.11 showSkeletons (iOS 11)

Xcode の Scene Editor 同様に、スキニングが適応されているジオメトリのボーンを表示されます。

Xcode 8 の Scene Editor は球体で表示されていましたが、3DCG の DCC ツールのように八面体を伸ばしたような形になりました。

### 53.2.12 設定例

複数設定が可能で、ARKit 使用時は SceneKit のデバッグオプションを併用することも可能です。

リスト 52.1: SCNDebugOptions

```
scnView.debugOptions = [
    .showPhysicsShapes,
    .showBoundingBoxes,
    .showLightInfluences,
    .showLightExtents,
    .showPhysicsFields,
    .showWireframe
]
```

# 第54章

## カスタムシェーダー

---

SceneKit でのカスタムシェーダーは主に 3 つです。

- SCNProgram でプリコンパイルしたものを使う
- SCNTechnique で設定し主にポストプロセスのように画面全体の変更する際に使う
- SCNShadable の shaderModifiers に Metal / GLSL のスニペットであるテキストデータを使う

SCNProgram と SCNTechnique は面倒、というかこの 2 つで本ができる難しさのため省略し、SCNShadable の shaderModifiers を軽く紹介します。shaderModifiers のシェーダーもそれなりに難しく、頻繁に使用するものでもないので、こういう機能があるということを覚えていればよいと思われます。

### 54.1 SCNShadable の **shaderModifiers** とは？

ノードのマテリアルに対して個別でカスタムシェーダーを適応することができます。カスタムシェーダーを適応する際は、定義された以下のエントリポイントを使用します。

エントリポイントは以下の 4 つとなり、上から実行されます。

エントリーポイント名	説明
geometry	ジオメトリ形状を変更する
surface	サーフェスプロパティ (Diffuse とか Material で設定しているもの) を変更する
lightingModel	ライト情報の適応する情報を変更する
fragment	全ての情報を計算後に色を変更する

## 54.2 使用方法

使い方としては SCNMaterial を初期化して、初期化したものに shaderModifiers の SCNShaderModifierEntryPoint ヘントリポイントを設定します。シェーダーの記述は String 型なので、テキストファイルを読み込む形にしても使用できます。

ちなみに、マテリアルの情報が書き換えられるため、ジオメトリが持つ firstMaterial は無視されます。もし、firstMaterial で変更したい場合は shaderModifiers で変更後設定してください。

また、Xcode 9 から Scene Editor でシェーダーの実行テストができるようになりました。シェーダーの変更を確認したい場合はこれが最も早いです。

## 54.3 設定例

適当なジオメトリに以降のコードを適応してください。

設定値はドキュメントより Scene Editor でシェーダー編集にあるヘルプボタンの方が値が列挙されていてわかりやすいです。

## 54.4 Geometry modifier

シーン自体の時間 (scn\_frame) でジオメトリの頂点を法線で伸ばし拡大するアニメーションをします。Xcode の Scene Editor ではフレーム再生されないので、Editor 下の再生ボタンを押してください。

リスト 53.1: Geometry modifier

```
let ballNode = SCNNNode(geometry: SCNSphere(radius: 2))
scene.rootNode.addChildNode(ballNode)

let geometryMaterial = SCNMaterial()
geometryMaterial.shaderModifiers = [
    SCNShaderModifierEntryPoint.geometry: "_geometry.position.xyz +=
        _geometry.normal * (sin(scn_frame.time) + 1);"
]

ballNode.geometry?.materials = [geometryMaterial]
```

## 54.5 Surface modifier

表面に白い帯状のものを塗ります。`_surface.diffuse` を掛け合わせているので、マテリアルにテクスチャを貼ると帯の部分だけ白くなります。

リスト 53.2: Surface modifier

```
let surfaceMaterial = SCNMaterial()
surfaceMaterial.shaderModifiers = [
    SCNShaderModifierEntryPoint.surface:
        """
        float Scale = 12.0;
        float Width = 0.25;
        float Blend = 0.3;
        vec2 position = fract(_surface.diffuseTexcoord * Scale);
        float f1 = clamp(position.y / Blend, 0.0, 1.0);
        float f2 = clamp((position.y - Width) / Blend, 0.0, 1.0);
        f1 = f1 * (1.0 - f2);
        f1 = f1 * f1 * 2.0 * (3. * 2. * f1);
        _surface.diffuse = _surface.diffuse + f1;
        """
]
ballNode.geometry?.materials = [surfaceMaterial]
ballNode.geometry?.firstMaterial?.diffuse.contents = UIColor.black
```

## 54.6 Lighting model modifier

ライトを編集する項目となり、以下のものはトゥーンシェーダーを模した表現をします。

リスト 53.3: Lighting model modifier

```
let ballNode = SCNNNode(geometry: SCNSphere(radius: 2))
scene.rootNode.addChildNode(ballNode)

let toonMaterial = SCNMaterial()
```

```

toonMaterial.shaderModifiers = [
    SCNShaderModifierEntryPoint.lightingModel:
    """
        vec3 lDir = normalize(vec3(0.1, 1.0, 1.0));
        float dotProduct = dot(_surface.normal, lDir);
        _lightingContribution.diffuse += (dotProduct *
        dotProduct * _light.intensity.rgb);
        _lightingContribution.diffuse = floor(_lightingContribution.diffuse *
        3.0) / 3.0;
    """
]
ballNode.geometry?.materials = [toonMaterial]
ballNode.geometry?.firstMaterial.diffuse.contents = UIColor.white

```

## 54.7 Fragment modifier

最終的に画像に落とし込むフラグメントシェーダーの処理を行うことができます。色に対して処理が行われるため、適当なテクスチャを貼っておくと結果がわかりやすいです。

下の例では、現状の色から 1 を引くことで色を反転させています。

リスト 53.4: Fragment modifier

```

let ballNode = SCNNNode(geometry: SCNSphere(radius: 2))
scene.rootNode.addChildNode(ballNode)

let fragmentMaterial = SCNMaterial()
fragmentMaterial.shaderModifiers = [
    SCNShaderModifierEntryPoint.fragment:
        "_output.color.rgb = vec3(1.0) - _output.color.rgb;"
]

ballNode.geometry?.materials = [fragmentMaterial]
ballNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named: "texture.png")

```

## 54.8 その他

シェーダーのスニペット内の変数の前に uniform を付けると、外部からそのマテリアルの状態を変更することができます。

あまり意味はないのですが、以下の例ではフラグメントシェーダーの処理によって、レンダリングされたジオメトリの画像の色を任意の数で明るくすることができます。

リスト 53.5: uniform

```
let ballNode = SCNNNode(geometry: SCNSphere(radius: 2))
scene.rootNode.addChildNode(ballNode)

let fragmentMaterial = SCNMaterial()
fragmentMaterial.shaderModifiers = [
    SCNShaderModifierEntryPoint.fragment:
    """
        uniform float val = 0.1;
        _output.color.rgb = _output.color.rgb * val;
    """
]
fragmentMaterial.setValue(2, forKey: "val")

ballNode.geometry?.materials = [fragmentMaterial]
ballNode.geometry?.firstMaterial?.diffuse.contents = UIImage(named: "texture.png")
```

# macOS アプリでのクリックイベント

macOS はタッチパネルでないので、iOS の場合は操作が異なります。

本書での iOS では Gesture Recognizer で設定していますので、それに対応する NSClickGestureRecognizer を実装します。

## iOS のタップイベント

リスト A.3:

```
let tapGesture = UITapGestureRecognizer(target: self, action:  
#selector(handleTap(_:)))  
scnView.addGestureRecognizer(tapGesture)
```

## macOS のタップイベント

リスト A.3:

```
let clickGesture = NSClickGestureRecognizer(target:  
self, action: #selector(handleClick(_:)))  
var gestureRecognizers = scnView.gestureRecognizers  
gestureRecognizers.insert(clickGesture, at: 0)  
scnView.gestureRecognizers = gestureRecognizers
```

タップやクリック後のセレクターの書き方は同じです。

リスト A.3:

```
@objc  
func handleTap(_ gestureRecognize: UIGestureRecognizer) {  
    // 内容  
}
```



## 編集後記

---

20年ぶりぐらいにちゃんした病気になつたり、ページの都合上、画像やコードが少なかつたりなどで、スキニング/ボーンやジョイント、SpriteKitを使用したHUDなどいくつかの機能を省略しています。

今回は新しい機能を紹介するためにこの本を作成しましたが、もし次回があるのなら、基本、アニメーション、エフェクトで分けた内容にしたり、3DCGのゲームをつくるまでの解説にしたり、Fox2などのAppleのサンプルデータの解説などの本もよいかもしてません。

# **SceneKit Book**

---

2017年10月22日 初版第1刷 発行

著 者 Toshihiro Goto

発行所 x67x6fx74fx6f

---

(C) 2017 Toshihiro Goto