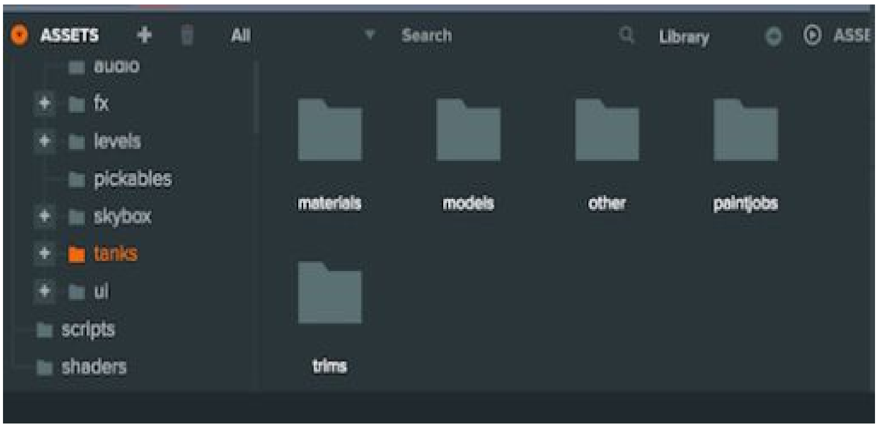
PlayCanvas unofficial

only about PlayCanvas

Autor: bunnymq

TOC

Part I - general



Chapter 1

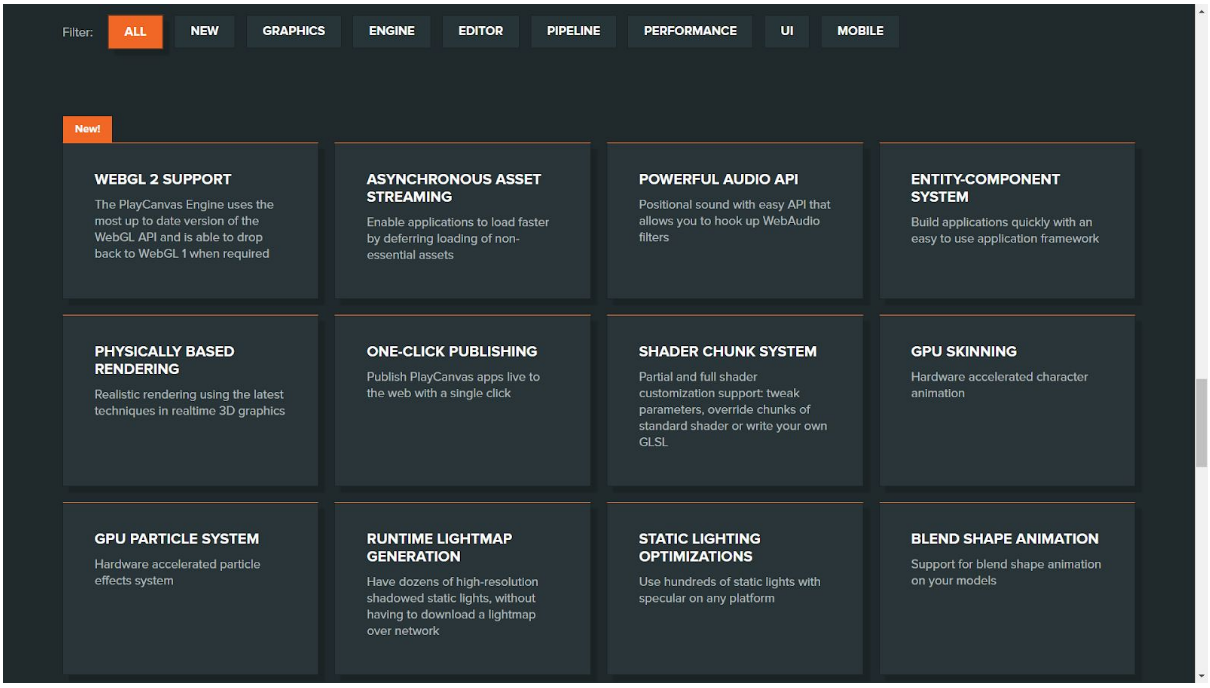
PlayCanvas, features

1.1 PlayCanvas - what is it?

A game engine created in WebGL, not based, not using 3D threejs library,
It was written from scratch, also as a cloud platform for creating games, visualizations, product configurators (e.g. car configurator). You can make very advanced graphics in it, thanks to the capabilities of the engine.
It has an integrated Ammo physics engine. You can use PlayCanvas as a platform with graphical editor and code editor in the cloud or as an engine-only, that is having the project locally on your laptop and using only the engine in the IDE or code editor of your choice (VS Code, Atom, Sublime Text), something like it is implemented in three.js.
With engine-only there is one advantage you can use git and upload to github, in the case of the platform you have to use PlayCanvas' version control system and checkpoints rather than commits like it works in git.
The name looks a bit like it's going to be about classic canvas, HTML5, which is 2D, but yet it's a rich 3D engine, a similar engine to PlayCanvas is Babylon.js, also worth a try, and PlayCanvas I recommend a really cool engine, but honestly only recently there was one drawback before, there was a resource space limit of 100MB if you used the platform. Now the limit is 1GB, so you can keep resources on the platform for free if the total does not exceed 1 GB, to have for example 10 GB or more you need to have a monthly subscription.
The community is not small, the documentation is well written, but there is one drawback: there are no books on the subject, neither in English nor in Polish.
In case of problems you can find a post concerning your problem or you can ask on the forum by creating a new post. It's not that nobody answers your question, you get an answer very fast and even a possible solution to your problem, which is a very strong advantage of PlayCanvas forum. I provide a link to the forum here PlayCanvas Discussion

Out of curiosity I looked through the engine code, it is really big, although not as huge as it is for Unreal Engine.

1.2 features



WEBGL 2 SUPPORT
The PlayCanvas Engine uses the most up to date version of the WebGL API and is able to drop back to WebGL 1 when required

ASYNCHRONOUS ASSET STREAMING
Enable applications to load faster by deferring loading of non-essential assets

POWERFUL AUDIO API
Positional sound with easy API that allows you to hook up WebAudio filters

ENTITY-COMPONENT SYSTEM
Build applications quickly with an easy to use application framework

PHYSICALLY BASED RENDERING
Realistic rendering using the latest techniques in realtime 3D graphics

ONE-CLICK PUBLISHING
Publish PlayCanvas apps live to the web with a single click

SHADER CHUNK SYSTEM
Partial and full shader customization support: tweak parameters, override chunks of standard shader or write your own GLSL

GPU SKINNING
Hardware accelerated character animation

GPU PARTICLE SYSTEM
Hardware accelerated particle effects system

RUNTIME LIGHTMAP GENERATION
Have dozens of high-resolution shadowed static lights, without having to download a lightmap over network

STATIC LIGHTING OPTIMIZATIONS
Use hundreds of static lights with specular on any platform

BLEND SHAPE ANIMATION
Support for blend shape animation on your models

PlayCanvas has the following features:

WebGL 2 support

Asynchronous resource streaming

audio API

ECS (Entity Component System) - about this in the Entity section

Physics-based rendering (PBR)

System chunk shader

GPU skinning

GPU particle system

Real-time light map generation

shape blending animation

soft shadows and light cookies

Resource importer and manager

Linear graphics pipeline and HDR

Input device API

SDF font renderer

rigidbody physics engine

tools for responsive interfaces

WebVR support

development and testing on a mobile device

resource filtering

real-time scene editing

cubic texture prefiltering

profiler

Texture compression (DXT, PVR and ETC1)

material editor

Cross-platform

WebGL 2 support

Engine uses the latest WebGL API, but is backward compatible with WebGL version one.

asynchronous streaming of resources

Asynchronous, and therefore faster loading of the application, by delaying the loading of less important resources.

audio API

Positional audio allows you to attach WebAudio filters.

ECS (Entity Component System) - about it in the Entity section

Create applications quickly using ECS.

physics-based rendering (PBR)

Bring realism to rendering with the latest real-time techniques in 3D graphics

shader chunk system

Partial and full shader customization: adjust parameters, overwrite standard shader chunks, or write your own GLSL code.

GPU skinning

Hardware accelerated character animation.

GPU particle system

Hardware accelerated particle system

Real-time light map generation

You can have multiple high resolution static lights

Shape blending animation

Support for shape blending animation of models

Soft shadows and light cookies

Choose from multiple shadow algorithms.

Light cookies provide cool effects at a cheap performance cost

asset importer and manager

Import assets: 3D models and animations (FBX, OBJ, DAE, 3DS), textures and HDR textures, audio files and more

linear and HDR graphics pipeline

Linear and HDR pipeline: gamma correction, tonemapping, support for HDR cubic textures and lightmaps

Input device API

Keyboard, mouse, gamepad, touchscreen support

SDF font renderer

Convert TTF, OTF to font resources (similar to Unity)

rigidbody physics engine

PlayCanvas' built-in Ammo physics system, which is a port of Bullet, allows for easier implementation of physics in the game

tools for responsive interfaces

Components to create responsive 2D and 3D interfaces

WebVR support

Support for the latest WebVR standards

Development and testing on a mobile device

Fast iterations using live updates on a mobile device

resource filtering

Search and filter your collection of assets

edit scenes in real time

Collaborate style changes on the fly with Google Docs

Cubic texture prefiltering

Set up image-based lighting (IBL) with just one click of a button

profiler

Displays graphs, real-time performance statistics

Texture compression (DXT, PVR, and ETC1)

One-click texture compression

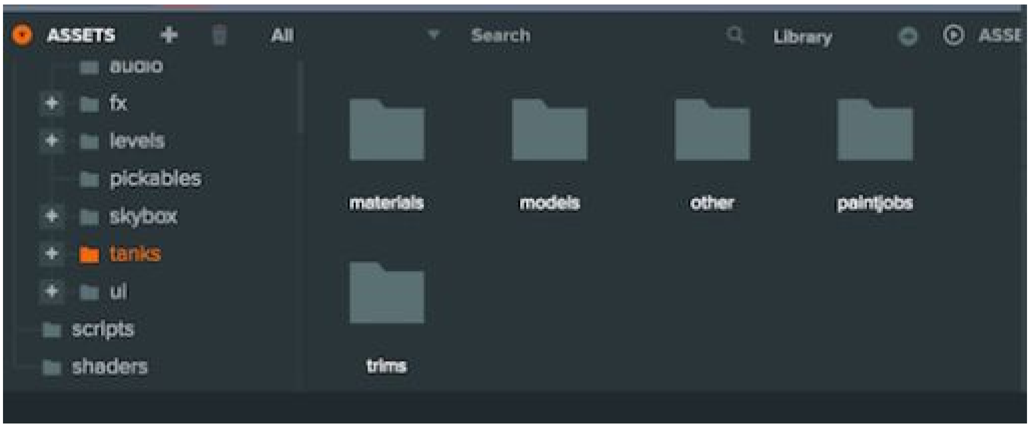
material editor

Quickly adjust visually visible changes to material parameters using the editor
multi-platform
Run the editor on any device: desktop, laptop, tablet, smartphone

As you can see PlayCanvas has a lot of functionality.
I will now move on to discuss the assets.

Chapter 2

assets



Assets can be of various types, e.g. model, animation, images for textures (.png,.jpg) and audio.

Below I discuss all types of resources available in PlayCanvas:

material

- Phong
- physical

texture

model

animation

cubemap texture

HTML

audio

CSS

shader

font
sprite
prefab (in pc under the name template)
Wasm module (Wasm module, WebAssembly module)

2.1 material

In general, the material defines surface properties such as color, gloss, etc. For exactly what this refers to please refer to your computer graphics textbooks.
In PlayCanvas, a material is one type of resource.
It has 2 subtypes: Phong and Physical.

Phong
The Phong shading model is an obsolete item, it is recommended to use the physical model. You can find more about the Phong shading model here Phong Material | Learn PlayCanvas
physical
Physical material represents an advanced, high quality shading model and is therefore recommended for use to achieve impressive results.
Detailed information about physical material properties is available here
Physical Material | Learn PlayCanvas

The following regions are related to this material: offset and tiling, ambient (related to ambient occlusion), diffuse (diffuse is also called albedo), specular (gives shine), emissive (emits light), opacity (transparency), normals (related to normal map), parallax (related to height map), environment (reflections), lightmap,

2.2 texture

A texture is an image that can be assigned to a material.
Below I have highlighted texture maps that are useful when multitexturing to get a more detailed look of the material.

Types of texture maps: ambient occlusion (AO map), cubemap, env map, diffuse map, specular map, emissive map, opacity map, normal map, height map, light map.
More about textures here Textures | Learn PlayCanvas

2.3 model

3D models and animations are created outside of PlayCanvas, exported from Blender, Wings3D, Maya or 3DS Max for example, and imported into PlayCanvas.
It is recommended to use the fbx format for best results and so the model will be converted to glb (i.e. fbx will remain as the source format, but glb will be created as the target format and thus there will be two fbx and glb formats for the model).

More about Models | Learn PlayCanvas

2.4 animation

The animation resource is used to play a single animation on a 3D model.
Full scene formats include animation, for example it is gltf, dae, fbx.

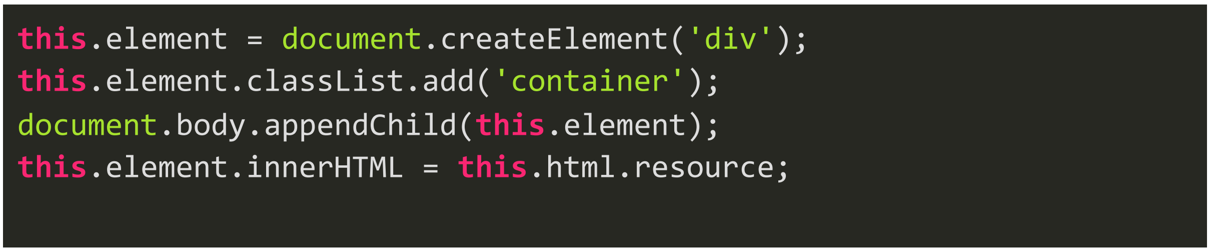
More about Animation | Learn PlayCanvas

2.5 cubemap texture

Cubic texture is a special texture type consisting of 6 texture resources.
It is used as skybox or environment map.
More about cubemap Cubemaps | Learn PlayCanvas

2.6 HTML

The HTML resource contains the HTML code.
To load HTML you need to write a piece of js code like this:



```
this.element = document.createElement('div');
this.element.classList.add('container');
document.body.appendChild(this.element);
this.element.innerHTML = this.html.resource;
```

Now I will quickly describe how the code works.

It creates a div element dynamically, then adds a class called container. Then it hooks that div to <body> and sets the content of your html as a child element for the container div.

This is one way to do it.

Of course you still have to add an attribute with html name (if you named it as this.html in the code, about attributes later), write html code, drag the html file in editor to a place where you can attach either entities or different resources, in this case it is ui under script component, in ui there is an attribute of resource type named html, only then you have HTML content on your page.

More about HTML

HTML | Learn PlayCanvas

2.7 audio

An audio resource is a sound file.

Audio | Learn PlayCanvas

2.8 CSS

A CSS resource contains the CSS code.

CSS style is attached to the page just as in the case of HTML resources, that is, you add an attribute named css, create CSS code, drag the css file in the editor to a place where you can attach either entities or different resources, for example, the ui under the script component, in the ui is just an attribute of the type of resource named css, only then you have the CSS content on your page, and so the applied appearance.

The code to hook up the CSS is a little different than it was with the HTML resource.

I'll show a slightly different way this time:

```
// get asset from registry by id
const asset = app.assets.get(32);

// create element
const style = pc.createStyle(asset.resource || '');
document.head.appendChild(style);

// when asset resource loads/changes,
// update html of element
asset.on('load', function() {
    style.innerHTML = asset.resource;
});

// make sure assets loads
app.assets.load(asset);
```

So this is how the CSS resource is fetched from the resource registry, the element is created, and the resource is loaded.

2.9 shader

The shader resource contains GLSL code, you can also upload files with the extension .vert, .frag or .glsl

```
const vertexShader = this.app.assets.find('my_vertex_shader');
const fragmentShader = this.app.assets.find('my_fragment_shader');
const shaderDefinition = {
    attributes: {
        aPosition: pc.SEMANTIC_POSITION,
        aUv0: pc.SEMANTIC_TEXCOORD0
    },
    vshader: vertexShader.resource,
    fshader: fragmentShader.resource
};
```

```
const shader = new pc.Shader(this.app.graphicsDevice, shaderDefinition);
const material = new pc.Material();
material.setShader(shader);
```

The first two lines are about looking for vertices and frags in the shader resource register. Next, a shader is defined with attributes: position and uv. The contents of the shader resource are appended to the vshader and fshader properties, first the vertex shader, then the fragment shader. As the penultimate step, the shader and material are created. Finally, the shader for the material is set.

2.10 font

A font resource contains an image with all the characters of the font, It is used to display text. More about font here Fonts | Learn PlayCanvas

2.11 sprite

A sprite is a 2D graphic, since the book is about creating a game in 3D, the 2D topic is omitted

More about sprite Sprite | Learn PlayCanvas

2.12 prefab (on pc under the name template)

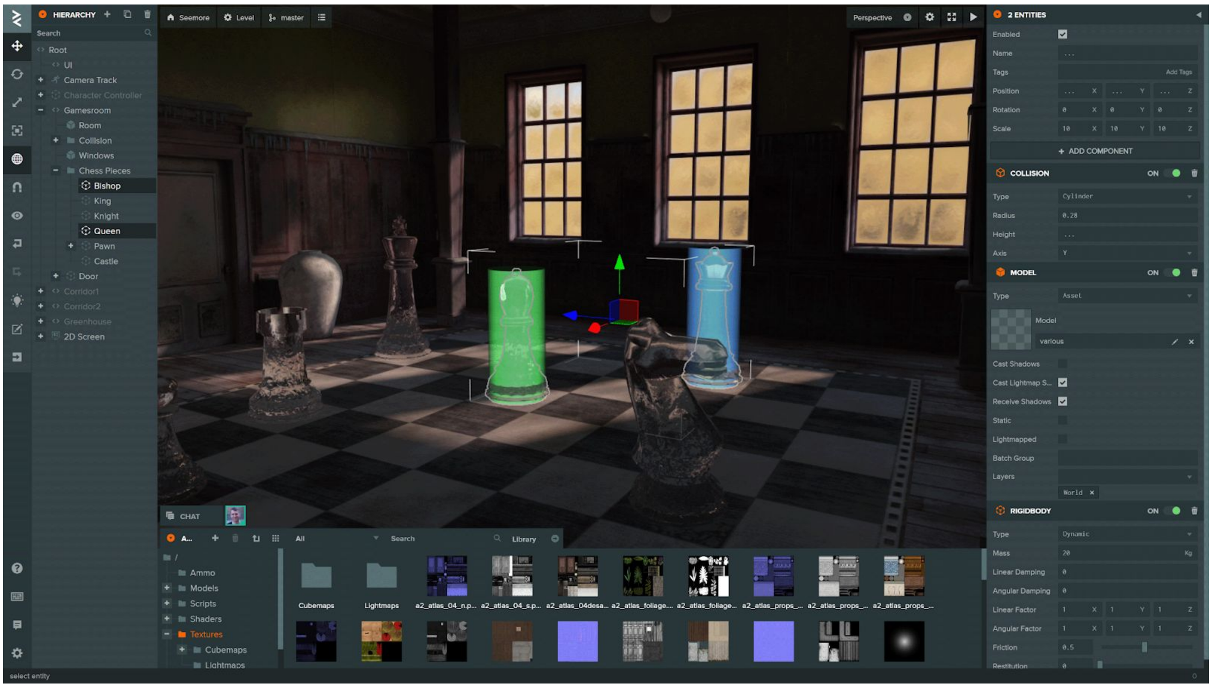
A prefab, which is a resource that contains a part of an entity, allows you to create multiple instances, so it is useful for constructing objects that look the same, e.g. 1000 trees of one type, 10 buttons that look the same, etc. In PlayCanvas, there are no prefab variants yet, i.e. for example there is a base prefab car, and for example I want to have different cars having the same features but different values, e.g. top speed or acceleration.

More about prefabs Template | Learn PlayCanvas

I skipped the topic of the Wasm resource. I think I've covered the possible resources in PlayCanvas pretty well.

Let me move on to the editor part.

Part II - editor



Enabled ☑
Name ...
Tags Add Tags
Position ... X ... Y ... Z
Rotation 0 X 0 Y 0 Z
Scale 10 X 10 Y 10 Z

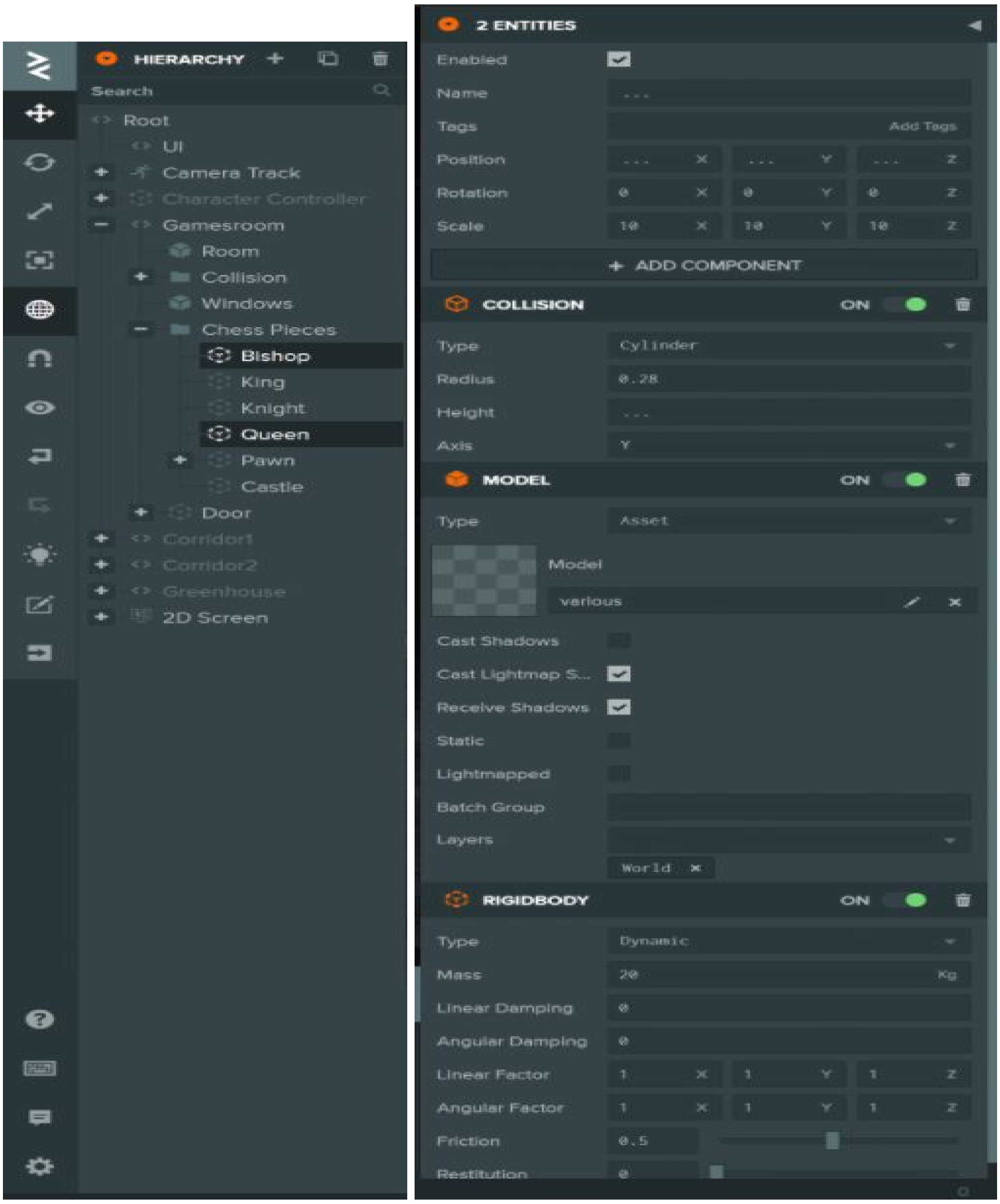
COLLISION ON
Type Cylinder
Radius 0.28
Height ...
Axis Y

MODEL ON
Type Asset
Model various
Cast Shadows □
Cast Lightmap S... ☑
Receive Shadows ☑
Static □
Lightmapped □
Batch Group
Layers World ×

RIGIDBODY ON
Type Dynamic
Mass 20 Kg
Linear Damping 0
Angular Damping 0
Linear Factor 1 X 1 Y 1 Z
Angular Factor 1 X 1 Y 1 Z
Friction 0.5
Restitution 0

Chapter 3

UI



Enabled ☑

Name ...

Tags Add Tags

Position ... X ... Y ... Z

Rotation 0 X 0 Y 0 Z

Scale 10 X 10 Y 10 Z

COLLISION ON

Type Cylinder

Radius 0.28

Height ...

Axis Y

MODEL ON

Type Asset

Model various

Cast Shadows □

Cast Lightmap S... ☑

Receive Shadows ☑

Static □

Lightmapped □

Batch Group

Layers World ×

RIGIDBODY ON

Type Dynamic

Mass 20 Kg

Linear Damping 0

Angular Damping 0

Linear Factor 1 X 1 Y 1 Z

Angular Factor 1 X 1 Y 1 Z

Friction 0.5

Restitution 0

3.1 Hierarchy panel

The hierarchy panel contains a scene graph, a tree view. This graph consists of a root, by default it is called Root, in this case it is called Main, it can also be called Game.

Main in this case is the parent of entities such as:

Camera, Board and Room Light, Board Folder, Dice1, Dice2, Tokens, Tile Owned, Houses, PropertyEntities, Walls, Cards, Colours, Furniture, New UI, Money, Property Cards, Property Lights and MainEntity.

This is a fragment of the board game Monopoly.

In addition, these entities are the parents (indicated by the plus sign) of other entities and so on.

As you can see, this structure is very complex, so it is important to group objects, e.g. as shown in the figure. Grouping objects is one of the good practices.

Hierarchical structure allows for good organization of game elements.

As a small digression: so look at how this is implemented in other examples of games created in PlayCanvas, go to the PlayCanvas website (you must be logged in to see EXPLORE content, once you are logged in go to explore, you will see various projects there, click on Project next to the particular project.

I chose SWOOP, it is an endless runner game.

This will take you to the next project overview

click on EDITOR,

click on the scene in this case Game and you can see the hierarchy.

I will show some other hierarchies
like the Space Buggy hierarchy

It's hard to capture on picture the whole developed hierarchy.

I will show one more hierarchy from Accelerally and I will end with hierarchies.

Some projects have locked Project option (e.g. TANX), you can only press PLAY to play the game , picture below.

3.2 Resources panel

Resources are best organized in folders, e.g. scripts in scripts, materials in materials, models in models, textures in textures etc.

On the left in the figure you can see the structure of the folders: / is the root and in it there are folders in this case: scripts, Chance, Community Chest, CSS, Furniture, HTML, Money, Other, Properties, Tokens, just like you have it organized in the file system on your operating system.

On the right you can see the folders and files, the folders mentioned above, 2 files: loading.js and redirect.js

Here you can upload your resources, you can filter by categories (here where All is), search for a resource (Search), add a new resource or delete an existing one, you can also enter the PlayCanvas Store.

3.3 Inspector panel

Here you can enable / disable an entity (Enabled), name the entity (Name)

Here you can enable / disable the entity (Enabled), name the entity (Name), add tags (Tags), set the transformation: position, orientation (rotation) and size (scale).

Importantly, all these properties are in local space, model space.

The orientation is set using so called Euler angles.

You can also add components (+ add component).

In this case, the added component is the script component which contains the ui.js code.

An entity can have many js scripts attached to it, such as UIHandler, Main, Money, Dice, Movement, Cards, PropertyLight, assets,

as shown in the figure.

So much for the inspector, there's still a menu and toolbar left to talk about.

I'll move on to the menus.

3.4 Menu Panel

The menu can be shown by clicking on the button with the PlayCanvas icon.

The menu lists all the commands you can do in the scene.

Here you can do the following things, add an entity, edit, start the game, get to help, view a list of available scenes, publish the game, burn a map of lights, open settings, set the priority of executing scripts.

Generally this is a shortcut if you can't find the button or can't remember the shortcut key.

3.5 Panel Toolbar

Panel toolbar contains the most common commands available in a convenient way.

The most useful is the run button (shortcut ctrl+enter), which starts the game in a new tab, loads the scene you are on and after loading you can test, play.

3.6 Viewport

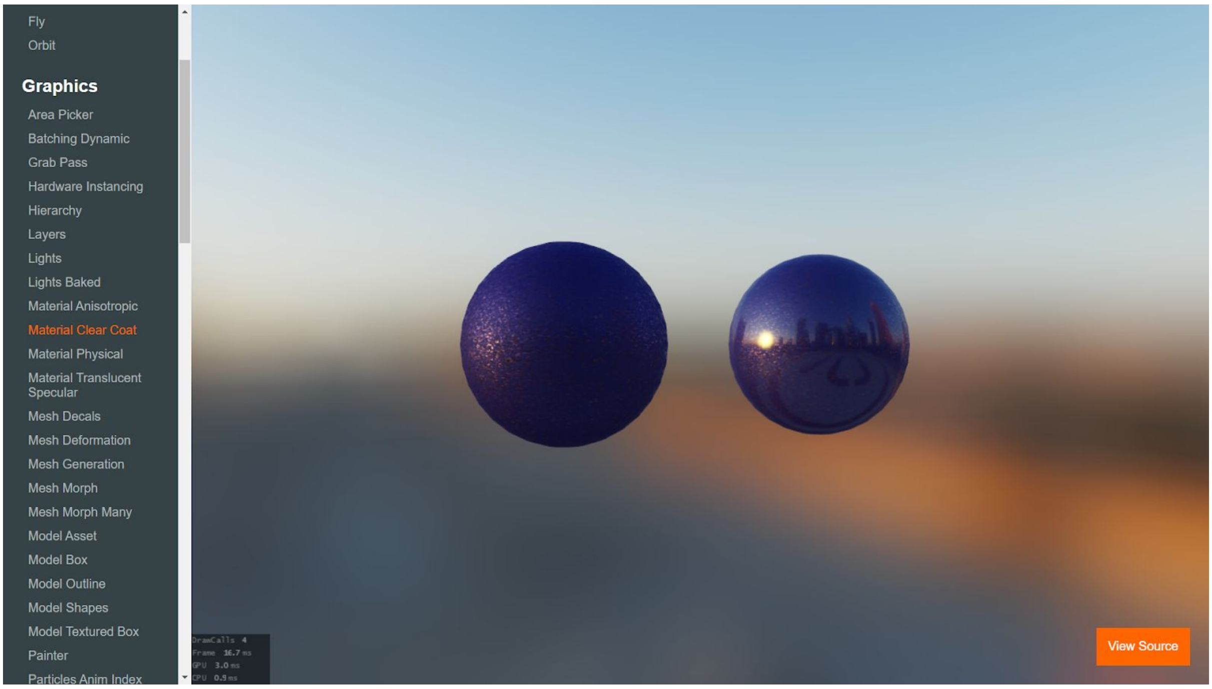
The viewport shows the scene currently displayed. You can move around the scene with the WASD keys and the up, down, left, right arrows. Holding shift accelerates the camera speed, you can view the scene faster this way if the space is large, such as here.

This is a city model, a mod for the game Assetto Corsa (pay no attention to the lack of textures), here is a very useful way to quickly view the scene.

Going back to the viewport you can set the camera to perspective or orthographic, in the case of ortho there are left, right, top, bottom, front, back views (left, right, top, bottom, front, back). You can still change to a view from another camera, such as a tracking camera (if you have set one in the hierarchy). In this example, the additional cameras are SplashCamera and Camera. As a side note, cameras are simply arrays.

Now I will move on to discuss the engine part.

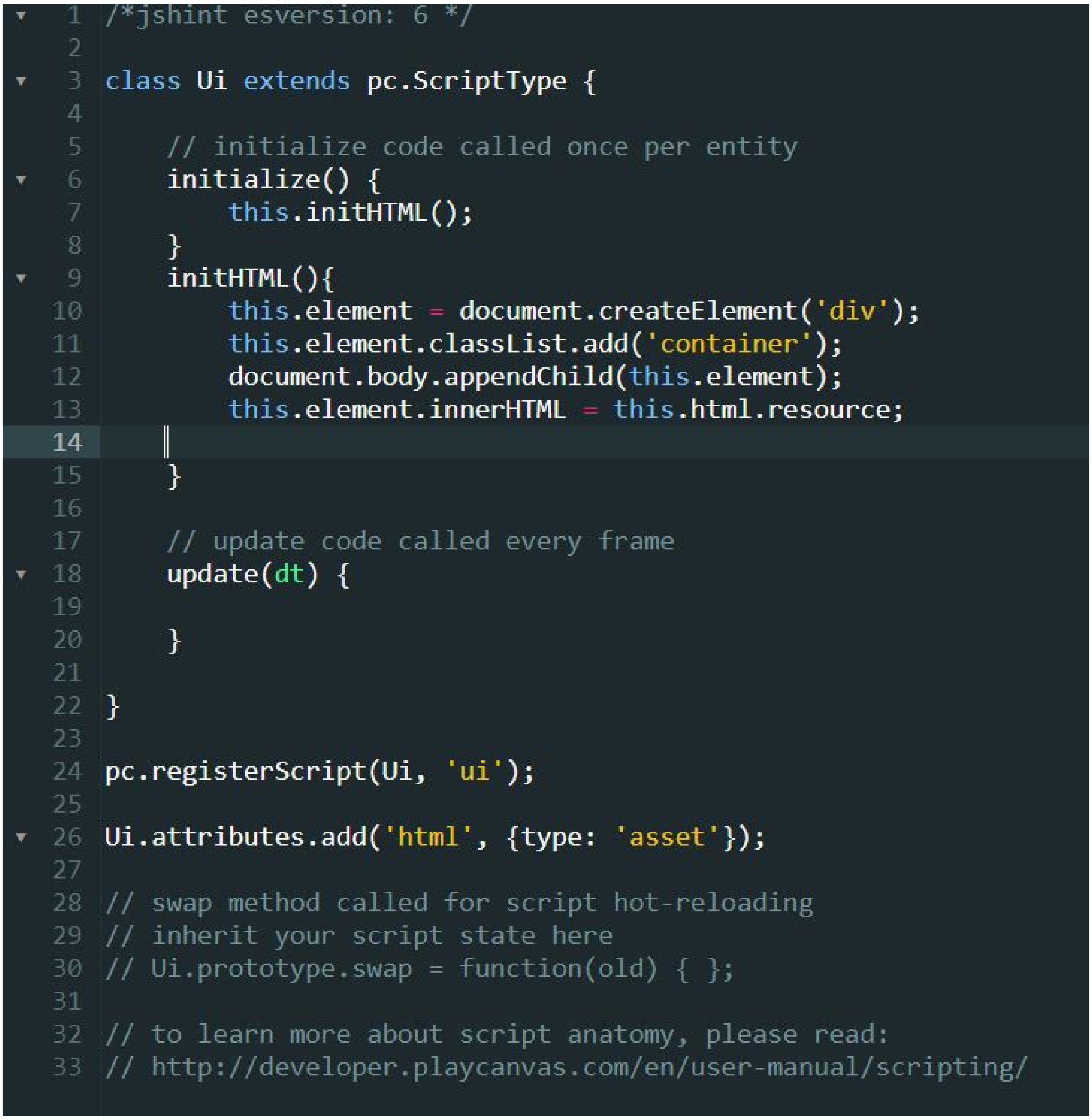
Part III - engine



```
DrawCalls  4
Frame  16.7 ms
GPU  3.0 ms
CPU  0.9 ms
```

Chapter 4

scripting



```
/*jshint esversion: 6 */

class Ui extends pc.ScriptType {

    // initialize code called once per entity
    initialize() {
        this.initHTML();
    }
    initHTML(){
        this.element = document.createElement('div');
        this.element.classList.add('container');
        document.body.appendChild(this.element);
        this.element.innerHTML = this.html.resource;

    }

    // update code called every frame
    update(dt) {

    }

}

pc.registerScript(Ui, 'ui');

Ui.attributes.add('html', {type: 'asset'});

// swap method called for script hot-reloading
// inherit your script state here
// Ui.prototype.swap = function(old) { };

// to learn more about script anatomy, please read:
// http://developer.playcanvas.com/en/user-manual/scripting/
```

3.1 initialize and update

initialize()

The initialize method refers to initialization that is performed only once for the entity to which the script has been added after the application loads. You use this method when you want to define variables and constants for a particular instance of the script,

e.g. this.speed, where this refers to the script, not to the window, and speed is variable name available in initialize and update, so you can pass this.speed variable from initialize to update.

update(dt)

Update can be implemented by using either time-based (time-based) or frame-based. If you do not use dt, which is a delta time (delta time) you are using frame-based animation. The result is a lack of fluidity in the animation.

On the other hand, if you add dt, then you get smooth animation based on time, not not frames per second (fps). That's why you usually multiply the parameter by dt, e.g. this.speed * dt.

3.2 attributes aka attributes.add()

With attributes you get the ability to iterate faster, i.e. you are able to experiment with parameters, create and test the game faster, e.g. if you are a designer and not a programmer, you can adjust parameters directly in the editor instead of in code. A concept similar to Unity.

You don't even need to use the dat.GUI.

There is no point in using attributes if you are engine-only, because you don't have access to the editor then.

Available attributes:

- entities
- resource
  - resource type
    - texture
    - model
    - material

color

curve

calculations

JSON

entities

One way even better than using find (find is generally a slow operation, maybe because it uses recursive depth-finding, DFS) is to refer to the entity outside the script using entity attribute. You give the entity name, its type, in the example below I refer to the car to have the information about the speed of the car in the ui. Let's assume that the car entity has a CarController script, then I can access the CarController using this car entity.

```
Ui.attributes.add('car', {type: 'entity'});
```

You can also use another way and not attach anything, that way is events which we will talk about later.

resource

- resource type
    - texture
    - model
    - material

color

curve

calculations

JSON

3.3 Communication - events

scriptA-scriptB events

application events

Chapter 5

Graphics

5.1 Camera

5.2 Lighting

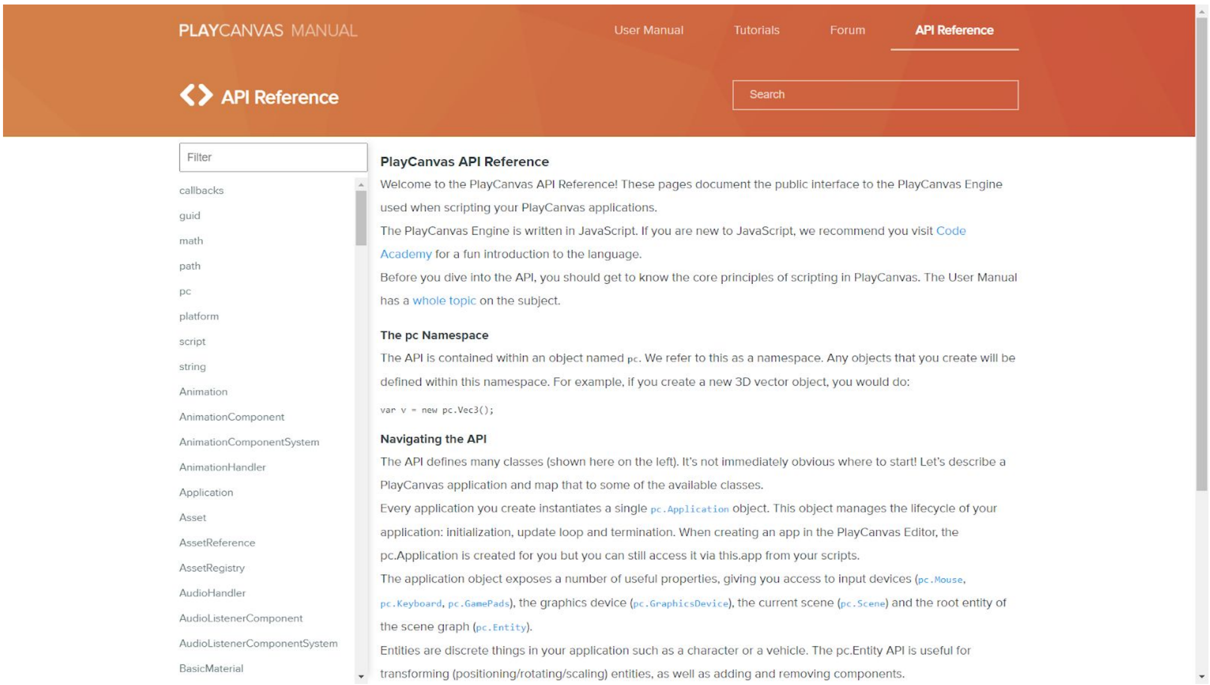
5.3 PBR

5.4 Particle system

5.5 Postprocessing

Chapter 6

API overview



PlayCanvas API Reference

Welcome to the PlayCanvas API Reference! These pages document the public interface to the PlayCanvas Engine used when scripting your PlayCanvas applications.

The PlayCanvas Engine is written in JavaScript. If you are new to JavaScript, we recommend you visit Code Academy for a fun introduction to the language.

Before you dive into the API, you should get to know the core principles of scripting in PlayCanvas. The User Manual has a whole topic on the subject.

The pc Namespace

The API is contained within an object named `pc`. We refer to this as a namespace. Any objects that you create will be defined within this namespace. For example, if you create a new 3D vector object, you would do:

```
var v = new pc.Vec3();
```

Navigating the API

The API defines many classes (shown here on the left). It's not immediately obvious where to start! Let's describe a PlayCanvas application and map that to some of the available classes.

Every application you create instantiates a single `pc.Application` object. This object manages the lifecycle of your application: initialization, update loop and termination. When creating an app in the PlayCanvas Editor, the pc.Application is created for you but you can still access it via this.app from your scripts.

The application object exposes a number of useful properties, giving you access to input devices (`pc.Mouse`, `pc.Keyboard`, `pc.GamePads`), the graphics device (`pc.GraphicsDevice`), the current scene (`pc.Scene`) and the root entity of the scene graph (`pc.Entity`).

Entities are discrete things in your application such as a character or a vehicle. The pc.Entity API is useful for transforming (positioning/rotating/scaling) entities, as well as adding and removing components.

Here we will analyze such classes as: Application, GraphNode, Entity.

There are also classes about the component, I will call it collectively xComponent (eg. AnimationComponent), where x is one of the given components:

Animation, Audio Listener, Button, Camera, Collision, Element, Layout Child, Layout Group, Light, Model, Particle System, Rigid Body, Screen, Script, Scrollbar, Scrollview, Sound, Sprite.

There are also systems of these components, or xComponentSystem, where x is one of the above components (e.g. AnimationComponentSystem).

Together, this structure (Entity + Component + ComponentSystem) creates something called ECS (Entity Component System). About components and component systems later.

And just like that I've gone through most of the PlayCanvas API, the rest of the classes can be found in Appendix A, and the most up to date list of classes can be found here PlayCanvas API Reference .
I will now move on to the Application class.

6.1 Application

Special attention should be paid to the basic app object of type Application. It is very specific in that it itself contains objects of type:
AssetRegistry, Scene, Keyboard, Mouse, Touch, Gamepad.
One more thing, pc is a namespace (short for PlayCanvas).

pc.app
assets - AssetRegistry
scene
root - Entity
graphicsDevice - graphic device (GraphicsDevice)

input:
keyboard Keyboard
mouse Mouse
touch - mobile device Touch
gamepad Gamepad

Other objects, properties, methods available here Application | PlayCanvas API Reference

6.2 GraphNode and Entity

6.2.1 GraphNode

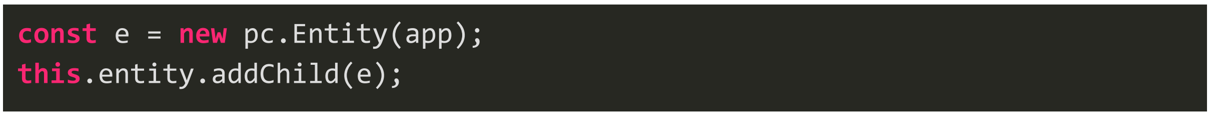
With the GraphNode class you can manipulate entities: add child entities,
retrieve / set orientation with Euler angles or quaternions, retrieve / set position in world or local space, retrieve / set scale only in local space, but also allow entity to look at a given point (lookAt()). There are a few other methods not covered in this class.

6.2.2 Entity

The Entity class extends / inherits from GraphNode, so it contains inherited methods such as: addChild(), get/set[Local]Position/Rotation(), getLocalScale(), lookAt(), get/set[Local]EulerAngles(), translate[Local](), rotate[Local]().

addChild(node)

Adds a child element, in this example it is an entity



```
const e = new pc.Entity(app);
this.entity.addChild(e);
```

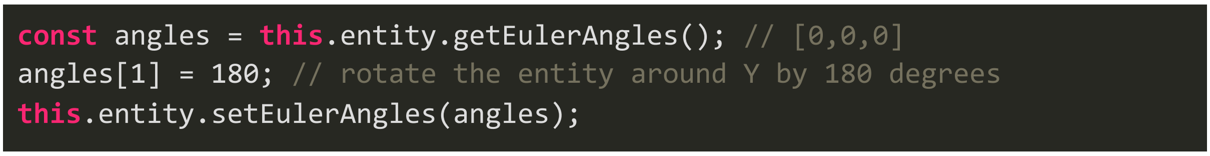
First an entity object is created and then this child element is added to the parent element (this.entity)

The most common way to set up a tracking camera, then the character/vehicle is the parent entity and the camera is the child entity. This is also a way to dynamically group objects, e.g. parent entity city, child entity building.

Position, orientation and scale can be set with three numbers x, y, z or one vector Vec3

getEulerAngles()

gets the Euler angles in world space



```
const angles = this.entity.getEulerAngles(); // [0,0,0]
angles[1] = 180; // rotate the entity around Y by 180 degrees
this.entity.setEulerAngles(angles);
```

As above, it retrieves to later rotate the entity around the Y axis by 180 degrees

getLocalEulerAngles()

takes Euler angles in local space

```
const angles = this.entity.getLocalEulerAngles();
angles[1] = 180;
this.entity.setLocalEulerAngles(angles);
```

gets to later rotate the entity around its own Y axis by 180 degrees

getLocalPosition()

gets the local position

```
const position = this.entity.getLocalPosition();
position[0] += 1; // move the entity 1 unit along x.
this.entity.setLocalPosition(position);
```

gets to later move the entity one unit along x.

getLocalRotation()

gets the local orientation

```
const rotation = this.entity.getLocalRotation();
```

getLocalScale()

takes a local scale

```
const scale = this.entity.getLocalScale();
scale.x = 100;
this.entity.setLocalScale(scale);
```

gets to set size x to 100 units later

getPosition()

gets a position in the world space

```
const position = this.entity.getPosition();
position.x = 10;
this.entity.setPosition(position);
```

gets to later set the x position to 10 units

getRotation()

gets orientation in world space

```
const rotation = this.entity.getRotation();
```

lookAt(x, [y], [z], [ux], [uy], [uz])

allows an entity to look at another entity, i.e. at a given point

```
// Look at another entity, using the (default) positive y-axis for up
const position = otherEntity.getPosition();
this.entity.lookAt(position);
```

```
// Look at the world space origin, using the (default) positive y-axis for up
this.entity.lookAt(0, 0, 0);
```

You can also set the entity to look at point 0, 0, 0

rotate(x, [y], [z])

Rotates an entity in world space

```
// Rotate via 3 numbers
this.entity.rotate(0, 90, 0);
// Rotate via vector
const r = new pc.Vec3(0, 90, 0);
this.entity.rotate(r);
```

rotateLocal(x, [y], [z])

Rotates an entity in local space

```
// Rotate via 3 numbers
this.entity.rotateLocal(0, 90, 0);
// Rotate via vector
const r = new pc.Vec3(0, 90, 0);
this.entity.rotateLocal(r);
```

setEulerAngles(x, [y], [z])

sets Euler angles in world space

```
// Set rotation of 90 degrees around world-space y-axis via 3 numbers
this.entity.setEulerAngles(0, 90, 0);
// Set rotation of 90 degrees around world-space y-axis via a vector
```

```
const angles = new pc.Vec3(0, 90, 0);
this.entity.setEulerAngles(angles);
```

setLocalEulerAngles(x, [y], [z])

sets Euler angles in local space

```
// Set rotation of 90 degrees around y-axis via 3 numbers
this.entity.setLocalEulerAngles(0, 90, 0);
// Set rotation of 90 degrees around y-axis via a vector
const angles = new pc.Vec3(0, 90, 0);
this.entity.setLocalEulerAngles(angles);
```

setLocalPosition(x, [y], [z])

sets the local position

```
// Set via 3 numbers
this.entity.setLocalPosition(0, 10, 0);
// Set via vector
const pos = new pc.Vec3(0, 10, 0);
this.entity.setLocalPosition(pos);
```

setLocalRotation(x, [y], [z], [w])

sets the local orientation

```
// Set via 4 numbers
this.entity.setLocalRotation(0, 0, 0, 1);
// Set via quaternion
const q = pc.Quat();
this.entity.setLocalRotation(q);
```

setLocalScale(x, [y], [z])

sets the local scale/size

```
// Set via 3 numbers
this.entity.setLocalScale(10, 10, 10);
// Set via vector
const scale = new pc.Vec3(10, 10, 10);
this.entity.setLocalScale(scale);
```

setPosition(x, [y], [z])

sets the position in world space

```
// Set via 3 numbers
this.entity.setPosition(0, 10, 0);
// Set via vector
const position = new pc.Vec3(0, 10, 0);
this.entity.setPosition(position);
```

setRotation(x, [y], [z], [w])

sets orientation in quaternions in world space

```
// Set via 4 numbers
this.entity.setRotation(0, 0, 0, 1);
// Set via quaternion
const q = pc.Quat();
this.entity.setRotation(q);
```

translate(x, [y], [z])

Moves an entity by x units in world space

```
// Translate via 3 numbers
```

```
this.entity.translate(10, 0, 0);
// Translate via vector
const t = new pc.Vec3(10, 0, 0);
this.entity.translate(t);
```

Here it moves 10 units in world space

translateLocal(x, [y], [z])

Moves an entity by x units in local space

```
// Translate via 3 numbers
this.entity.translateLocal(10, 0, 0);
// Translate via vector
const t = new pc.Vec3(10, 0, 0);
this.entity.translateLocal(t);
```

Here it moves 10 units in local space

Part IV - Entity - game object

Chapter 7
Entity, Component, System

Entity

Components (18)

- Animation
- Audio Listener
- Button
- Camera
- Collision
- Element
- Layout Child
- Layout Group
- Light
- Model
- Particle System
- Rigid Body
- Screen
- Script
- Scrollbar
- Scrollview
- Sound
- Sprite

Systemy

Part V - examples

Chapter 8 PlayCanvas - examples

Two technical demos will be analyzed, After the Flood and Casino,

I will not discuss the code in detail, because it would take hundreds or even thousands of pages, so I will analyze the demos in terms of hierarchy (and in great detail, not as briefly as it was in the part about the editor in the hierarchy panel),

organization of resources, especially scripts, but also in terms of graphics.

8.1 After the Flood

Hierarchy analysis

Analysis of resources

Graphics analysis

8.2 Casino

Hierarchy analysis

Analysis of resources

Graphics analysis

Appendix A

API

pc

callbacks

guid

math

path

platform

script

string

Animation

Animation

AnimationComponent

AnimationComponentSystem

AnimationHandler

Asset

Asset

AssetReference

AssetRegistry

Audio

AudioHandler

AudioListenerComponent

AudioListenerComponentSystem

Batch

Batch

BatchGroup

BatchManager

Component

Component

ComponentSystem

ComponentSystemRegistry

Element

ElementComponent

ElementComponentSystem

ElementDragHelper

ElementInput

ElementInputEvent

ElementMouseEvent

ElementTouchEvent

Layout

LayoutChildComponent

LayoutChildComponentSystem

LayoutGroupComponent

LayoutGroupComponentSystem

Model

Model

ModelComponent

ModelComponentSystem

ModelHandler

Morph

Morph

MorphInstance

MorphTarget

Script

ScriptAttributes

ScriptComponent

ScriptComponentSystem

ScriptHandler

ScriptRegistry

ScriptType

Sound

Sound

SoundComponent

SoundComponentSystem

SoundInstance

SoundInstance3d

SoundManager

SoundSlot

Sprite

Sprite

SpriteAnimationClip

SpriteComponent

SpriteComponentSystem

SpriteHandler

Texture

Texture

TextureAtlas

TextureAtlasHandler

TextureHandler

Touch

Touch

TouchDevice

TouchEvent

Vec

Vec2

Vec3

Vec4

Vertex

VertexBuffer

VertexFormat

VertexIterator

XR

XrHitTest

XrHitTestSource

XrInput

XrInputSource

XrManager

pozostałe

Application

BasicMaterial

bounding

BoundingBox

BoundingSphere

button

ButtonComponent

ButtonComponentSystem

camera

CameraComponent

CameraComponentSystem

collision

CollisionComponent

CollisionComponentSystem

Color

contact

ContactPoint

ContactResult

container

ContainerHandler

ContainerResource

Controller

CubemapHandler

curve

Curve

CurveSet

Entity

EventHandler

font

Font

FontHandler

ForwardRenderer

Frustum

GamePads

GraphicsDevice

GraphNode

Http

I18n

IndexBuffer

keyboard

Keyboard

KeyboardEvent

layer

Layer

LayerComposition

light

LightComponent

LightComponentSystem

Lightmapper

Mat

Mat3

Mat4

Material

Material

MaterialHandler

Mesh

Mesh

MeshInstance

Mouse

Mouse

MouseEvent

Node

OrientedBox

particle system

ParticleSystemComponent

ParticleSystemComponentSystem

Picker

post effect

PostEffect

PostEffectQueue

Quat

ray

Ray

RaycastResult

RenderTarget

resource

ResourceHandler

ResourceLoader

rigidbody

RigidBodyComponent

RigidBodyComponentSystem

scene

Scene

SceneHandler

scope

ScopeId

ScopeSpace

screen

ScreenComponent

ScreenComponentSystem

scrollbar

ScrollbarComponent

ScrollbarComponentSystem

scrollview

ScrollViewComponent

ScrollViewComponentSystem

Shader

SingleContactResult

Skeleton

skin

Skin

SkinInstance

StandardMaterial

StencilParameters

Tags

TransformFeedback

stałe

Appendix B

PlayCanvas Examples

Animacja

- Blend
- Tweening

Kamera

- First Person
- Fly
- Orbit

Grafika

- Area Picker
- Batching Dynamic
- Grab Pass
- Hardware Instancing
- Hierarchy
- Layers
- Lights
- Lights Baked
- Material Anisotropic
- Material Clear Coat
- Material Physical
- Material Translucent Specular
- Mesh Decals
- Mesh Deformation
- Mesh Generation
- Mesh Morph
- Mesh Morph Many
- Model Asset

- Model Box
- Model Outline
- Model Shapes
- Model Textured Box
- Painter
- Particles Anim Index
- Particles Random Sprites
- Particles Snow
- Particles Sparks
- Point Cloud
- Point Cloud Simulation
- Portal
- Post Effects
- Render To Cubemap
- Render To Texture
- Shader Burn
- Shader Toon
- Shader Wobble
- Texture Basis
- Transform Feedback

Loadery

- Loader Glb
- Loader Obj

urządzenia wejścia

- Gamepad
- Keyboard
- Mouse

Różne

- Mini Stats
- Multi Application

Fizyka

- Compound Collision
- Falling Shapes

- Raycast
- Vehicle

Dźwięk

- Positional

Spine

- Alien
- Dragon
- Goblins
- Hero
- Spineboy

interfejs użytkownika

- Button Basic
- Button Particle
- Button Sprite
- Scroll View
- Text Basic
- Text Canvas Font
- Text Drop Shadow
- Text Localization
- Text Markup
- Text Outline
- Text Typewriter
- Text Wrap
- Various

mieszana rzeczywistość (vr, ar)

- Ar Basic
- Ar Hit Test
- Vr Basic
- Vr Controllers
- Vr Hands
- Vr Movement
- Xr Picking