Table Of Content

What is «2D/3D Paint»?

«2D/3D Paint» - is an asset for Unity that allows you to paint on 2D and 3D objects! You can also create modern paint app with cool features and great performance!

To use asset you need to add a prefab, one component and configure a few parameters! With «2D/3D Paint» you will be able to paint on 2D and 3D components such as: MeshRenderer, SkinnedMeshRenderer, SpriteRenderer and RawImage.

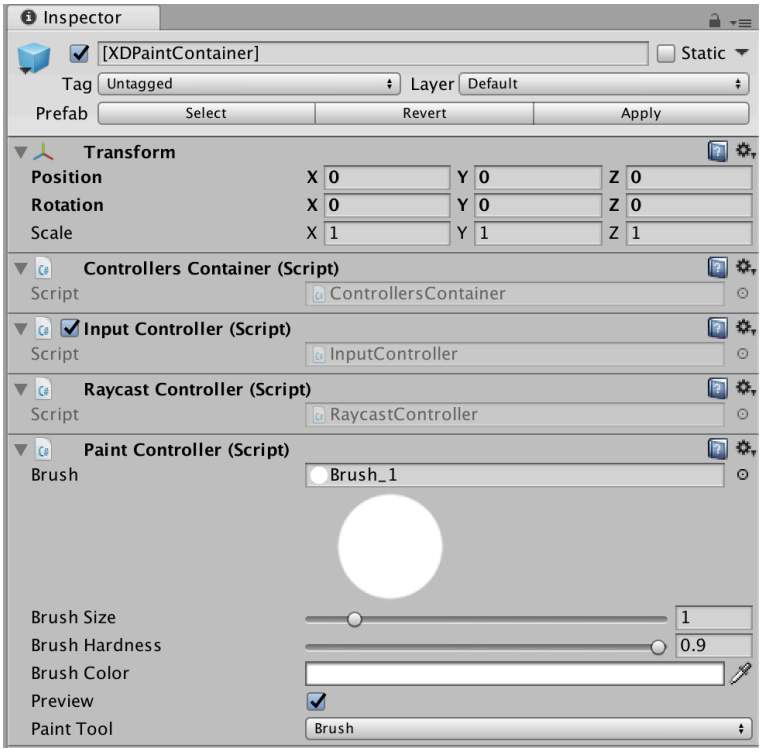
Requirements

For correct work «2D/3D Paint» requires:

- Unity 2017.4 or newer;
- GameObject with supported components such as MeshRenderer, SkinnedMeshRenderer, SpriteRenderer or RawImage with material. Also when you are using the MeshRenderer or SkinnedMeshRenderer component, make sure that their models have UV map.

Quick Start

Add a prefab to the scene from path “Assets/XDPaint/Prefabs/[XDPaintContainer]”. This prefab contains singleton-components, let’s look at parameters of PaintController component:



- **Brush** - brush texture. Make sure that «Wrap Mode» in the settings of the brush texture equals «Clamp»;

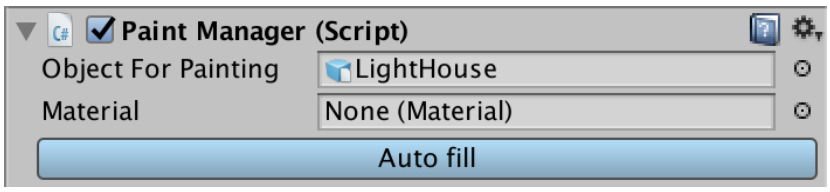
- **Brush Size** - brush size;
- **Brush Hardness** - brush hardness;
- **Brush Color** - brush color;
- **Preview** - to show preview of the brush while user hovers over the object to be painted;
- **Paint Tool** - selected tool. Supported tools:
    - Brush - brush tool for painting on the texture;
    - Erase - erase tool;
    - Eyedropper - eyedropper tool for picking a color of the brush;
    - BrushSampler - tool for sampling brush texture.

After that, create the GameObject with the **PaintManager** component using the Unity menu «GameObject -> 2D/3D Paint». Select the GameObject to paint in the «Object For Painting» field. The object to paint must contain one of the supported components:

- MeshRenderer
- SkinnedMeshRenderer
- SpriteRenderer
- RawImage

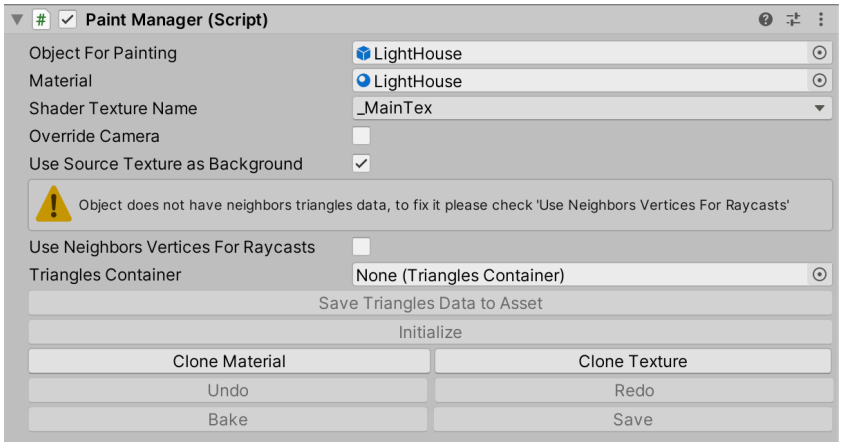
If any child of **PaintManager** GameObject contains an object with one of the supported components, it can automatically assign it to the «Object For Painting» field by clicking on the «Auto fill» button:

After clicking on the «Auto fill» button, the «Object For Painting» field will be filled in when one of the supported components will be found:

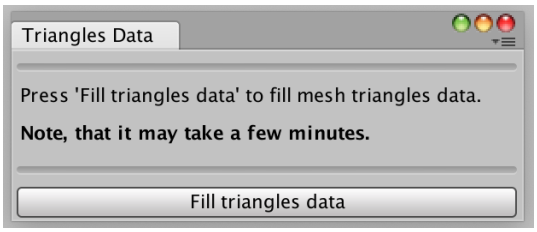


After that, in the same way assign the material to paint «Material» by repeatedly pressing the «Auto fill» button. If the material or painting object cannot be found automatically, make sure that there is a GameObject with the supported component among the child objects, otherwise «Object For Painting» and «Material» can be assigned manually.

Other component settings will appear if «Object For Painting» and «Material» have values. Consider the existing settings:



- **Shader Texture Name** - the name of the material texture on which the painting will be performed;
- **Override Camera** - asset overrides the camera to determine the intersection of the ray with triangles and to work with user input. If the flag is set to false, the camera is obtained from the Camera.main;
- **Camera** - camera for determining ray intersections with the triangles and for working with user input;
- **Use Source Texture as Background** - use source texture as a background layer of paint result;
- **Use Neighbors Vertices For Raycasts** - asset uses neighbors vertices to find the intersection of the ray with triangles while lines are drawing. If flag is unchecked the results of calculations can be inaccurate with non-convex objects. When we are using objects with large number of the vertices the performance of searching the intersection of the ray with triangles is degrading. It is recommended to set the flag as true. After setting a flag will be opened a window with confirmation:

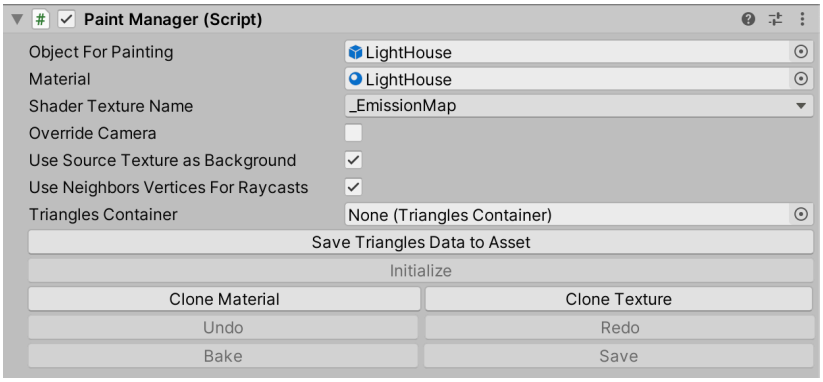


After pressing “Fill triangles data” will be showed progress bar with filling progress. Wait until it finish searching all neighbour triangles, then windows will be closed and drawing lines work properly. User can save generated triangles data to ScriptableObject using “Save Triangles Data to Asset” Button;

- **Triangles Container** - link to TrianglesContainer ScriptableObject, may be empty - in that case triangles data will be stored in component field;
- **Clone Material** - button to copy the source material of the object to new file, can be used only in the Unity Editor;
- **Clone Texture** - button to copy the source texture of the object to the new file, can be used only in the Unity Editor;
- **Undo** - button to undo action with the object, can be used only in the Unity Editor;

- **Redo** - button to redo action with the object, can be used only in the Unity Editor;
- **Bake** - button to save the painting results to the source file. Note that texture is stored in RAM and is not written to disk, can be used only in the Unity Editor;
- **Save** - button to save the texture of the painting results to the file, can be used only in the Unity Editor.

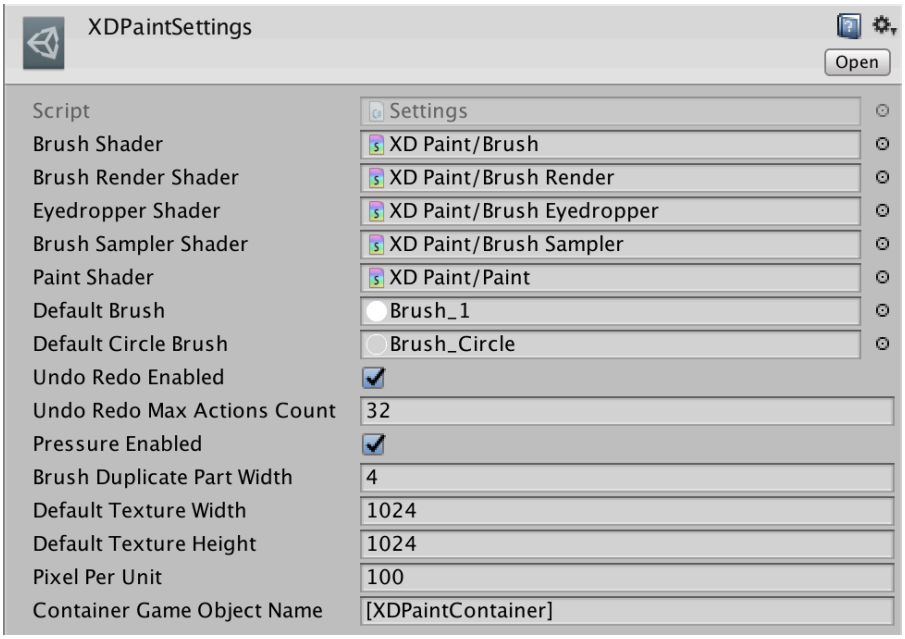
Let's set up the object to paint on the emission texture:



Texture of Shader with the name «_EmissionMap» is selected. After switching to the game mode, the user will be able to paint on the object «LightHouse». The material and the source texture «_EmissionMap» will be cloned, RenderTexture will be assigned as the new texture. User can use input devices for painting: a computer mouse or a touch device.

Settings

«2D/3D Paint» has global settings. The configuration file is located at: Assets/XDPaint/Resources/XDPaintSettings.asset. The settings file is a ScriptableObject with fields. Consider the fields of the settings file:



- **Default Brush** - default brush texture;

- **Default Circle Brush** - default brush texture for BrushSampler tool;
- **Undo Redo Enabled** - undo and redo functionality is enabled;
- **Undo Redo Max Actions Count** - maximum amount of actions that stores Undo/Redo;
- **Pressure Enabled** - pressure force is applied to the brush size;
- **Brush Duplicate Part Width** - the width of the duplicated part of the brush, the value affects the number of vertices and smoothness of the line while lines are drawing. The higher the value, the less the number of vertices will be drawn;
- **Default Texture Width** - the default texture width is using for objects that do not have the source texture;
- **Default Texture Height** - the default texture height is using for objects that do not have the source texture;
- **Pixel Per Unit** - pixelPerUnit field for sprites. It is using for objects that do not have the source sprite;
- **Container Game Object Name** - the name of the GameObject, for the container object with InputController and RaycastController components;

Tools

With version 2.0 assets support tools. Tool - is an instrument for processing texture when user paint on it. Asset has 4 built-in tools: brush, erase, eyedropper and brush sampler. User can switch between this tools for get different drawing result. Let's look at tools functional:

- **Brush** - default tool for painting;
- **Erase** - erase drawing result;
- **Eyedropper** - pick up color and set it as a color of brush;
- **BrushSampler** - copying a part of drawing area to new texture.

Brushes

With version 2.0 assets has global parameters for brushes. They can be set in PaintController component:

- **Texture** - brush texture;
- **Size** - size of the brush;
- **Hardness** - the hardness setting using for rounding brush;
- **Color** - color of the brush;
- **Opacity** - opacity value of the brush.

How it works

General description

«2D/3D Paint» clones the source material and replaces the source texture with the RenderTexture in which the painting takes place. Before that, the source texture is copied in the RenderTexture. The asset passes data to (BasePaintObject) using a class to handle user input(InputController). The UV coordinate is calculated in (BasePaintObject) to determine the position of the painting on the texture. Painting takes place on the previously created RenderTexture and is stored in GPU memory, which provides high performance.

Painting

«2D/3D Paint» creates a RenderTexture(in ARGB32 format) texture. The size of the RenderTexture is equal to the size of the source texture. If there is no source texture, the size for RenderTexture will be taken from the settings. For such objects as MeshRenderer and SkinnedMeshRederer, the asset uses ray-surface intersection to determine the intersection of the model triangle with the ray. The use of models with large number of vertices can lead to loss of performance. To solve this problem was implemented the finding neighbors triangles.
«2D/3D Paint» creates two RenderTexture for each paint object - PaintTexture and CombinedTexture.

The PaintTexture is using for painting, source texture is written to the CombinedTexture, and then the PaintTexture is written.

Brush

Brush is a texture, that renders into RenderTexture using brush parameters. When user changes brush parameters like Texture, Opacity, Color or Hardness, brush invokes Render method that renders brush into RenderTexture.

Undo/redo

«2D/3D Paint» stores painting textures and save them every OnMouseUp event. It can be turned off or limited using Settings.

Input

«2D/3D Paint» processes input using the InputController class. Input can be with the mouse or by touching the screen.

Raycast

For MeshRenderer and SkinnedMeshRenderer objects finding the intersection of the ray with the triangle is used to calculate the UV coordinates. These calculations are performed on

the CPU, the time of which depends on the number of vertices of the model. In order to avoid high CPU loads, it should be noted that the more vertices the model contains, the more CPU time it will take to find the intersection.

To optimise the calculations and drawing lines was added the method based on the data on neighbors triangles. To draw a line from triangle «A» to triangle «B» asset uses neighbors triangles data to find the entry and exit positions of the triangles. This method significantly improves performance for objects with large number of vertices, but can draw a line with inaccuracy for non-convex objects, because it does not check data about other triangles that may lie «closer» to the camera and close the verifiable(neighboring) vertices.

API Help

Content

PaintManager script

The script is a manager for paint object. It combines the main script for painting on object, contains instances of Paint, BasePaintObject.

Main public fields, properties and methods:

public GameObject ObjectForPainting - GameObject of the object to be painted;
public bool ShouldOverrideCamera - overrides the camera;
public BasePaintObject PaintObject { ... } - property of the painted object;
public Camera Camera { ... } - property of the camera;
public bool UseSourceTextureAsBackground { ... } - property of using source texture as background texture for resulting image;

public bool UseNeighborsVerticesForRaycasts { ... } - property of using neighbors vertices for raycasts when drawing lines;
public bool HasTrianglesData { ... } - whether component contains any triangles data;
public bool Initialized { ... } - property of the initialization status of the object;
public void Init() - initializes PaintManager. If PaintManger was initialized before, it will re-create its internal data: RenderTextures, Meshes and Materials;
public void Render() - invokes object rendering;
public void FillTrianglesData(bool fillNeighbors = true) - fills model data, argument - to fill data about neighbors triangles;
public void ClearTrianglesData() - removes filled information about triangles;
public void ClearTrianglesNeighborsData() - removes filled information about the neighboring triangles;
public void GetTriangles() - returns triangles array;
public void SetTriangles(Triangle[] trianglesData) - sets triangles data directly from code using array of Triangles;
public void SetTriangles(TrianglesContainer trianglesContainerData) - sets triangles data directly from code using TrianglesContainer ScriptableObject;
public RenderTexture GetPaintTexture() - returns RenderTexture of painting;
public RenderTexture GetResultRenderTexture() - returns result RenderTexture (source texture + painting texture + preview);
public Texture2D GetResultTexture() - returns the resulting texture;
public void Bake() - writes changes to the source texture, used in Unity Editor.
public void UpdatePreviewInput() - updates events for preview mode(used by PaintController).

BasePaintObject class

Base class for painting on RenderTexture. It can be declared as **CanvasRendererPaint**, **MeshRendererPaint**, or **SpriteRendererPaint**. The derived classes **CanvasRendererPaint**, **MeshRendererPaint**, and **SpriteRendererPaint** contain logic to check the painting position based on the data from the InputController and return the UV texture position for further work according to the base class logic.

Main public fields, properties and methods:

public event PaintDataHandler OnPaintDataHandler - painting event, can be used by the developer to obtain data about painting;
public event PaintHandler OnPaintHandler - painting lines event, used by ToolsManager;
public event PaintHandler OnMouseHoverHandler - mouse hover event, used by ToolsManager;
public event PaintHandler OnMouseDownHandler - mouse down event, used by ToolsManager;
public event PaintHandler OnMouseHandler - mouse press event, used by ToolsManager;
public event PaintHandler OnMouseUpHandler - mouse up event, used by ToolsManager;
public event PaintHandler OnUndoHandler - undo action event, used by ToolsManager;
public event PaintHandler OnRedoHandler - redo action event, used by ToolsManager;
public bool IsPainting { ... } - property, whether user is painting;
public bool ProcessInput - whether input processing for current paint object;

public bool UseSourceTextureAsBackground - use source texture as background of paint texture;
public new Camera Camera { ... } - camera property used to determine the position of the painting on the texture;
public TextureKeeper TextureKeeper - an instance of the class that stores data about painting states, contains painting textures, used to undo and redo actions;
public void Init(Camera camera, Transform objectTransform, Paint paint, IRenderTextureHelper renderTextureHelper) - initialization of the object;
public void Destroy() - destroys previously created RenderTextures and Meshes;
public void OnMouseHover(Vector3 position, Triangle triangle = null) - on mouse hover method;
public void OnMouseDown(Vector3 position, float pressure = 1f, Triangle triangle = null) - on mouse down method;
public void OnMouseButton(Vector3 position, float pressure = 1f, Triangle triangle = null) - on mouse button method;
public void OnMouseUp() - on mouse up method;
public void DrawPoint(Vector2 position, float pressure = 1f) - draws point from code using texture position;
public void DrawLine(Vector2 positionStart, Vector2 positionEnd, float pressureStart = 1f, float pressureEnd = 1f) - draws line from code using texture positions;
public void FinishPainting() - force end painting;
public void OnRender() - renders to RenderTexture;
public void RenderCombined() - renders to RenderTexture, in the case when using two RenderTexture's (PaintManager.renderTextureMode as RenderTexturesMode.TwoTextures);
public void ClearTexture() - clears the texture to paint, to display changed data you must call the render method using Render().

TextureKeeper class

A class for storing and managing stored textures. It contains paint textures and is used for undo and redo operations.

Main public methods:

public void Init(Action<int, int, Dictionary<int, State>> extraDraw, bool enabled) - initialization of the object, the first argument - action for rendering according to the passed data, the second argument - whether functionality is enabled;
public void Undo() - undo the action;
public void Redo() - redo the action;
public void Reset() - deletes all stored textures for undo and redo;
public bool CanUndo() - checks if undo is possible;
public bool CanRedo() - checks if redo is possible.

Paint class

A class for storing and managing data about painting material and its parameters.

Basic public properties and methods:

public Material Material { ... } - painting material;
public void Init(IRenderComponentsHelper renderComponentsHelper) - painting material initialization;
public void Destroy() - destroys previously created Materials;
public void SetObjectMaterialTexture(Texture texture) - sets new texture of the object;
public void SetPreviewTexture(Texture texture) - sets preview texture;
public void SetPaintTexture(Texture texture) - sets paint texture to material;
public void SetPaintPreviewVector(Vector4 brushOffset) - sets data to display the painting preview.

PaintController script

Script-singleton that stores all PaintManagers, ToolsManager and Brush with its parameters. Used to set/get a tool, brush and parameters.

Main public fields, properties and methods:

public ToolsManager ToolsManager { ... } - returns ToolsManager;
public PaintTool Tool { ... } - current tool;
public bool Preview { ... } - brush preview;
public Brush Brush { ... } - returns Brush class;
public void Init(PaintManager paintManager) - initializing PaintManager;
public PaintManager[] ActivePaintManagers() - returns active PaintManagers.

Brush class

A class for storing and managing data about brush material and its parameters.

Basic public properties and methods:

public Material Material { ... } - brush material;
public Color Color { ... } - brush color;
public Texture SourceTexture { ... } - source texture of the brush;
public RenderTexture RenderTexture { ... } - renders texture of the brush;
public float Size { ... } - brush size;
public float Hardness { ... } - brush hardness;
public ChangeColorHandler OnChangeColor; - event of changing the brush color;
public ChangeTextureHandler OnChangeTexture; - event of changing the texture of the brush;
public void Init() - initializing the brush material;
public void Destroy() - destroys previously created RenderTexture, Mesh and Material;
public void SetColor(Color colorValue, bool render = true, bool sendToEvent = true) - sets the brush color;
public void SetTexture(Texture texture, bool render = true, bool sendToEvent = true) - sets the brush texture;

public void SetPaintTool(PaintTool paintTool) - sets current tools for changing shader params.

ToolsManager class

Class for managing tools, contains all tools and manages them.

Main public fields, properties and methods:

public BasePaintTool CurrentTool - returns current tool;
public void Init(PaintManager paintManager) - initializing for PaintManager, subscribe for PaintManager events;
public void SetTool(PaintTool paintTool) - sets tool.

BasePaintTool class

Base class for every tool. Can process image using input events from PaintManager.

Main public fields, properties and methods:

public virtual PaintTool Type { ... } - tool type;
public virtual bool ShowPreview { ... } - whether to show preview or not;
public virtual bool RenderToPaintTexture { ... } - whether to render to PaintTexture or not;
public virtual bool AllowRender { ... } - whether to allow any render or not;
public virtual bool RenderToTextures { ... } - whether to render to any RenderTextures;
public virtual bool DrawPostProcess { ... } - whether to draw post process data or not;
public virtual void Enter() - enter to the tool;
public virtual void Exit() - exit from the tool;
public virtual void UpdateHover(Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseHover;
public virtual void UpdateDown(Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseDown;
public virtual void UpdatePress(Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseButton;
public virtual void OnPaint(Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnPaintPointOrLine;
public virtual void UpdateUp(Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseUp;
public virtual void OnDrawPostProcess(Vector2 uv, Vector2 paintPosition, float pressure) - invokes after finished combining textures in current frame;
public virtual void OnUndo(object sender) - invokes on BasePaintObject.TextureKeeper.OnUndo;
public virtual void OnRedo(object sender) - invokes on BasePaintObject.TextureKeeper.OnRedo.

InputController script

Script-singleton is a component for user input management.

Main public fields, properties and methods:

public event OnInputPositionHit OnMouseHover - mouse hover event on objects with SpriteRenderer and RawImage;
public event OnInputPositionHit OnMouseHoverWithHit - mouse hover event on objects with MeshRenderer and SkinnedMeshRederer;
public event OnInputPositionHitPressure OnMouseDown - left mouse click event on objects with SpriteRenderer and RawImage;
public event OnInputPositionHitPressure OnMouseDownWithHit - event of pressing the left mouse on the objects with MeshRenderer and SkinnedMeshRederer;
public event OnInputPositionHitPressure OnMouseButton - left mouse button pressing event on objects with SpriteRenderer and RawImage;
public event OnInputPositionHitPressure OnMouseButtonWithHit - left mouse button pressing event on objects with MeshRenderer and SkinnedMeshRederer;
public event OnInputPosition OnMouseUp - event of releasing the left key of the mouse;
public Camera Camera { ... } - camera property used for user input and raycasts.

RaycastController script

Script-singleton is a controller for data checks of ray intersections with the triangles and the keeper of the data about all available objects to make a Raycast checks.

Main public fields, properties and methods:

public Camera Camera { ... } - property of the camera, used for raycasts;
public void InitObject(Camera newCamera, Component paintComponent, Component renderComponent, Triangle[] triangles) - initializes a new mesh object;
public void DestroyMeshData(Component renderComponent) - destroys previously created mesh data for raycasts;
public void Raycast(Ray ray, out Triangle triangle) - checks the intersection of the ray with the triangles objects, the result will be returned to out Triangle triangle;
public void RaycastLocal(Ray ray, Transform objectTransform, out Triangle triangle) - checks the intersection of the ray with the triangles of the Transform objectTransform, the result will be returned to out Triangle triangle;
public void NeighborsRaycast(Triangle triangle, Ray ray, out Triangle outTriangle) - checks the intersection of the ray with the neighboring triangles of Triangle triangle, the result will be returned to out Triangle outTriangle;

Frequently used methods

Check if there is ability to undo/redo:

```
PaintManager.PaintObject.TextureKeeper.CanUndo();
PaintManager.PaintObject.TextureKeeper.CanRedo();
```

Undo/Redo action:

```
PaintManager.PaintObject.TextureKeeper.Undo();
PaintManager.PaintObject.TextureKeeper.Redo();
```

Remove undo/redo data:

```
PaintManager.PaintObject.TextureKeeper.Reset();
```

Repaint object:

```
PaintManager.Render();
```

Clear render texture:

```
PaintManager.PaintObject.ClearTexture();
PaintManager.Render();
```

Change tool:

```
PaintController.Instance.Tool = PaintTool…;
```

Brush size:

```
PaintController.Instance.Brush.Size = value;
```

Brush hardness:

```
PaintController.Instance.Brush.Hardness = value;
```

Change brush color:

```
var brushColor = PaintController.Instance.Brush.Color;
brushColor = new Color(color.r, color.g, color.b, brushColor.a);
PaintController.Instance.Brush.SetColor(brushColor);
```

Change brush alpha:

```
var color = PaintController.Instance.Brush.Color;
color.a = value;
PaintController.Instance.Brush.SetColor(color);
```

Enable/Disable input for all PaintManagers:

```
InputController.Instance.enabled = value;
```

Enable/Disable input for some PaintManager:

```
PaintManager.PaintObject.ProcessInput = value;
```

Drawing from code

Asset supports drawing from code. For drawing in texture space invoke one of this methods:

`PaintManager.PaintObject.DrawPoint(new Vector3(512, 512));` - drawing a point, where argument - position on texture;

```
PaintManager.PaintObject.DrawLine(new Vector2(400, 612), new Vector2(100, 100));
```
- drawing a line, where arguments - start and end position on texture;

For drawing in screen-space for MeshRenderer/SkinnedMeshRenderer components need to invoke methods:

```
Vector3 screenPos = new Vector3(...);
var ray = Camera.main.ScreenPointToRay(screenPos);
RaycastController.Instance.Raycast(ray, out triangle);
```

then use functions for painting:

```
PaintManager.PaintObject.OnMouseDown(screenPos, 1f, triangle);
PaintManager.PaintObject.OnMouseButton(screenPos, 1f, triangle);
PaintManager.PaintObject.OnMouseUp(screenPos, 1f, triangle);
```

For drawing in screen-space for SpriteRenderer/RawImage components need to invoke methods for painting:

```
Vector3 screenPos = new Vector3(…);
```

then use functions for painting:

```
PaintManager.PaintObject.OnMouseDown(screenPos);
PaintManager.PaintObject.OnMouseButton(screenPos);
PaintManager.PaintObject.OnMouseUp(screenPos);
```

Creating PaintManager from code

Paint manager can be created from code. This is an example of creating PaintManager using method:

```
public void AddPaintManager(GameObject objectForPainting, Material material, TrianglesContainer triangles)
    {
     var paintManager = objectForPainting.AddComponent<PaintManager>();
     paintManager.ObjectForPainting = objectForPainting;
     paintManager.Material.SourceMaterial = material;
     paintManager.Material.ShaderTextureName = "_MainTex";
     if (objectForPainting.GetComponent<MeshRenderer>() != null || objectForPainting.GetComponent<SkinnedMeshRenderer>() != null)
     {
          paintManager.SetTriangles(triangles);
     }
     paintManager.Init();
    }
```

User can re-initialize PaintManager with one line of code. Note that after it, all previously created resources (RenderTextures, Meshes, Materials), will be re-created:

```
_paintManager.Init();
```

User can change texture for painting using code:

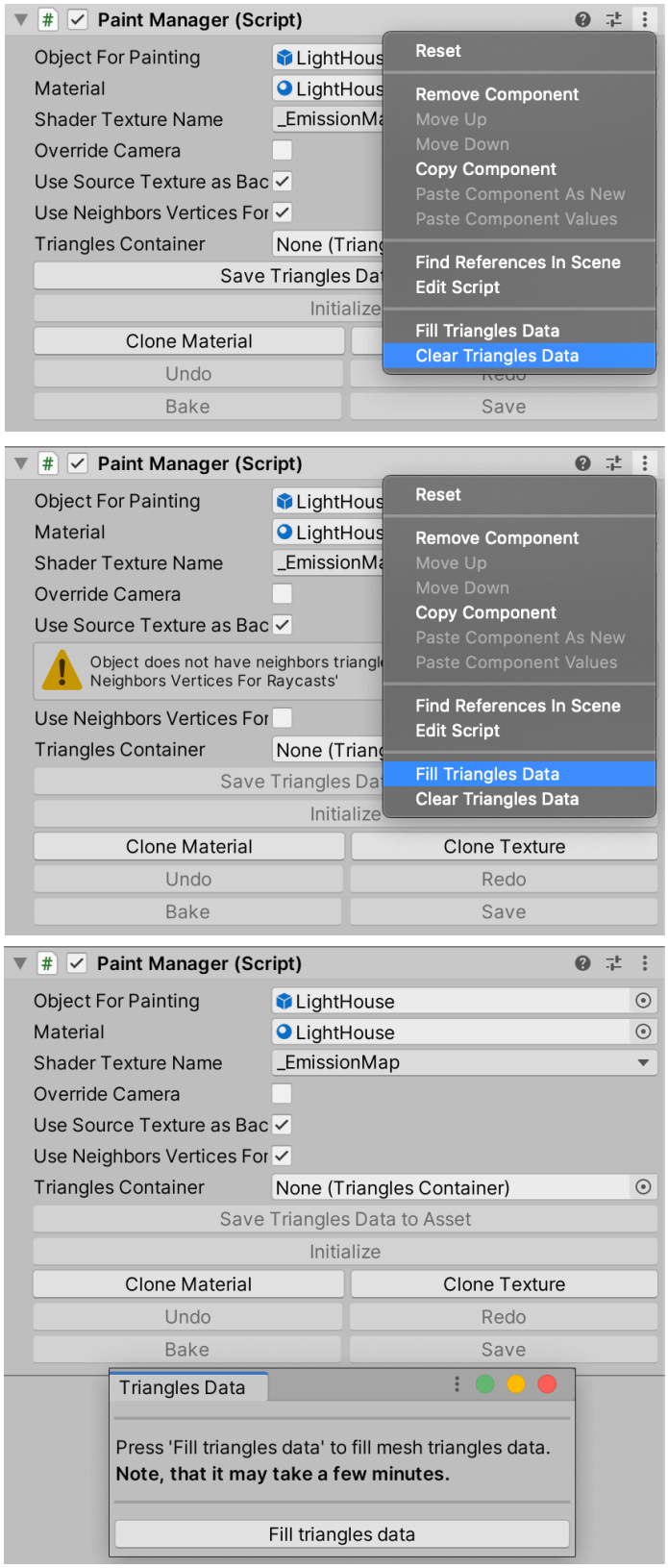
```
public void ChangeTexture(Texture texture)
{
 var material = _paintManager.Material.SourceMaterial;
 material.SetTexture(_paintManager.Material.ShaderTextureName,
texture);
 _paintManager.Material.SourceMaterial = material;
 _paintManager.Init();
 }
```

Tips

Here are tips which will help in its configuration and use «2D/3D Paint»:

- Using a custom brush, make sure that their textures «Wrap Mode» are set as «Clamp» in order to avoid duplication of brushes with preview mode on;
- Use the «Use Neighbors Vertices For Raycasts» for **PaintManager** if it is possible. Using this flag allows you to draw lines for the objects such as MeshRenderer and SkinnedMeshRederer using less CPU time, but there may be inaccuracies with painting on non-convex objects. Please note that generation of data may take a few seconds, it depends on the count of vertices;
- Field «Triangles» of **PaintManager** contains data about the vertices of the model. Data can take up disk space and contain the indices of vertices, the number(ID) of the triangle and the indices of the vertices of the neighboring triangles. Use the context menu to add and/or delete data. If the flag «Use Neighbors Vertices For Raycasts» is set to false, then there is no need to use the data of neighboring triangles and this data will be deleted. This field does not show by Inspectors in Unity Editor, but can be viewed using Debug mode of Inspector window of Unity Editor;
- Field «Override Camera» for **PaintManager** sets the camera for the objects **InputController** and **RaycastController**, which are singleton-objects. Thus, it should be understood that the «Camera» field in **PaintManager** can overwrite the field values of singleton objects in the case when two or more **PaintManager** with different camera settings are used at the same time;
- «2D/3D Paint» supports Unity Lightweight Render Pipeline, for LWRP use LWRP-compatible shaders for object for painting;
- Demo with SkinnedMeshRenderer can be downloaded here, just import *.unitypackage after importing «2D/3D Paint» from AssetStore;

- After opening asset in Unity 2019+ assets serialization format can be changed and triangles data might be corrupt. It's highly recommended to clear triangles data for every PaintObject that works with MeshRenderer and SkinnedMeshRenderer and add triangles data again:



Please let me know if you have any questions.
E-mail: unitymedved@gmail.com