Table of Contents

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.

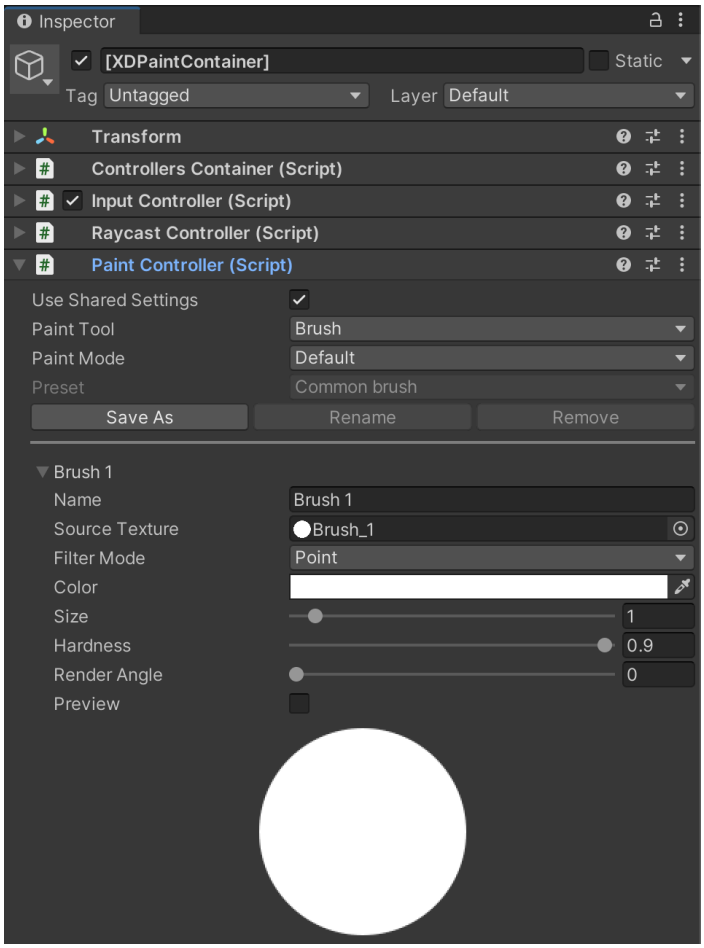
Requirements

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

- Unity 2018.4 or newer;
- GameObject with supported component: MeshRenderer, SkinnedMeshRenderer, SpriteRenderer or RawImage with material. Also when you use the MeshRenderer or SkinnedMeshRenderer component, make sure that their model has 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:

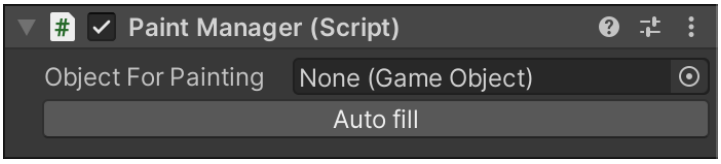


- **Use Shared Settings** - whether to use settings of PaintController (Brush, Paint Tool, Paint Mode, and Preview) for all PaintManagers;
- **Paint Tool** - selected tool. Supported tools:
    - Brush - brush tool for painting on the texture;
    - Erase - erase tool;
    - Eyedropper - eyedropper tool for picking a color of the brush;
    - BrushSampler - tool for sampling brush texture;
    - Clone - tool for cloning parts of the texture;
    - Blur - tool for blurring parts of the texture;
    - GaussianBlur - tool for blurring parts of the texture.
- **Paint Mode** - mode of painting, can be Default or Additive. Default mode bakes draw result into Paint Texture each frame, Additive mode provides more accurate color and alpha blending, bakes draw result in Paint Texture on Mouse Up event;
- **Preset** - brush preset that saved in Assets/XDPaint/Resources/XDPaintBrushPresets.asset. Can be used for saving brushes with different parameters and easily switching between them. See Brush and Presets paragraph for details.
- Brush Parameters:
    - **Name** - name of the brush, must be unique;
    - **Source Texture** - brush texture. Make sure that «Wrap Mode» equals «Clamp» in the settings of the brush texture;
    - **Filter Mode** - FilterMode for brush RenderTexture;
    - **Color** - brush color;
    - **Size** - brush size;
    - **Hardness** - brush hardness. Rounds brush when values less than 1;
    - **Render Angle** - brush render angle in degrees;
    - **Preview** - to show preview of the brush while user hovers over the object to be painted;

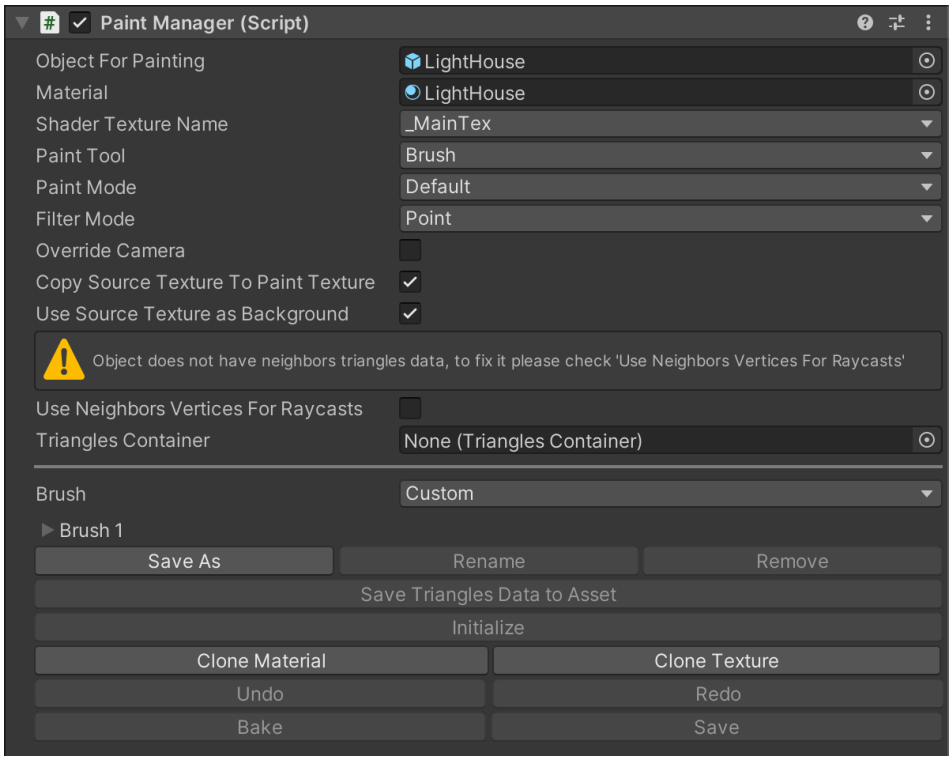
After that, create the GameObject with the **PaintManager** component using the Unity menu «GameObject -> 2D/3D Paint». Select the GameObject to paint in the «Object For Painting» field. The object to paint must contain one of the supported components:

- 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:

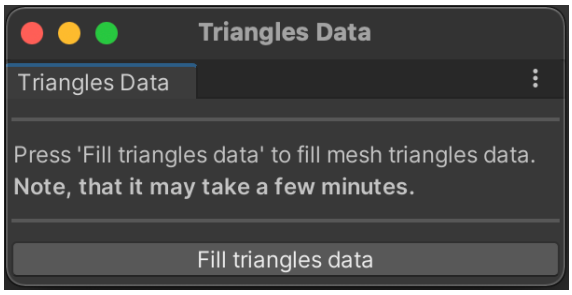


After that, in the same way assign the material to paint «Material» by repeatedly pressing the «Auto fill» button. If the material or painting object cannot be found automatically, make sure that there is a GameObject with the supported component among the child objects, otherwise «Object For Painting» 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;
- **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;
- **Paint Tool** - painting tool, if PaintController has checked flag “Use Shared Settings”, use PaintController to set shared Paint Tool for all PaintManagers;
- **Paint Mode** - mode of painting: default or additive;
- **Filter Mode** - FilterMode for painting RenderTextures;
- **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;

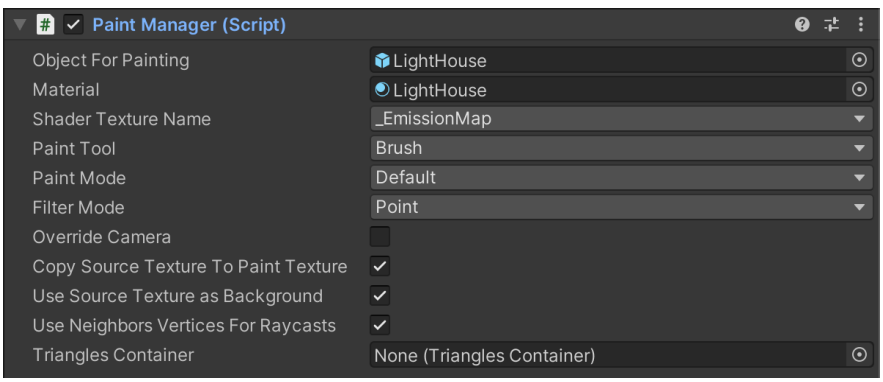
- **Use Neighbors Vertices For Raycasts** - asset uses neighbors vertices to find the intersection of the ray with triangles while lines are drawing. If flag is unchecked the results of calculations can be inaccurate with non-convex objects. When we are using objects with large number of the vertices 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 a window opens with confirmation:



After pressing “Fill triangles data” progress bar shows 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;

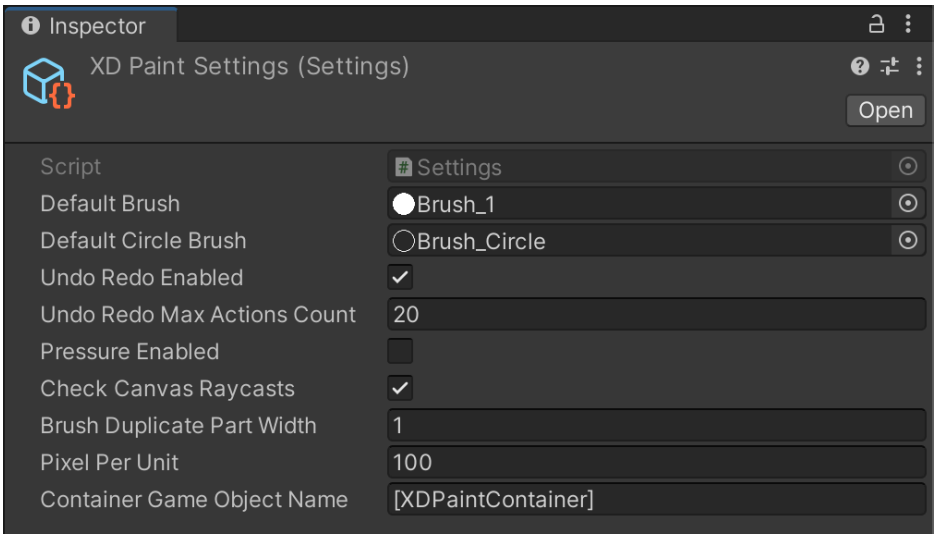
- **Triangles Container** - link to TrianglesContainer ScriptableObject, may be empty - in that case triangles data will be stored in component field;
- **Brush** - brush parameters. When selected “Custom” - PaintManager has unique brush parameters. User can select ready-to-use preset of the brush from list. Note that if PaintController has flag “Use Shared Settings”, brush parameters will be used from PaintController;
- **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/Clone tools;
- **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;
- **Check Canvas Raycasts** - prevent painting on component if any other canvas components lies on over painting object. This also require setting InputController settings: Canvas and Ignore For Raycasts (optional) fields;
- **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;
- **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

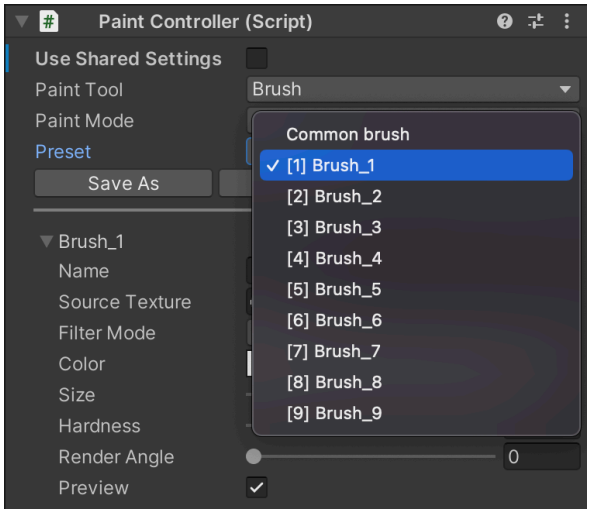
From version 2.0 assets support tools. Tool - is an instrument for processing texture when user paint on it. Asset has 6 built-in tools: brush, erase, eyedropper, brush sampler, clone, blur tool and gaussian blur tool. User can switch between this tools for get different drawing result. Let's look at tools functional:

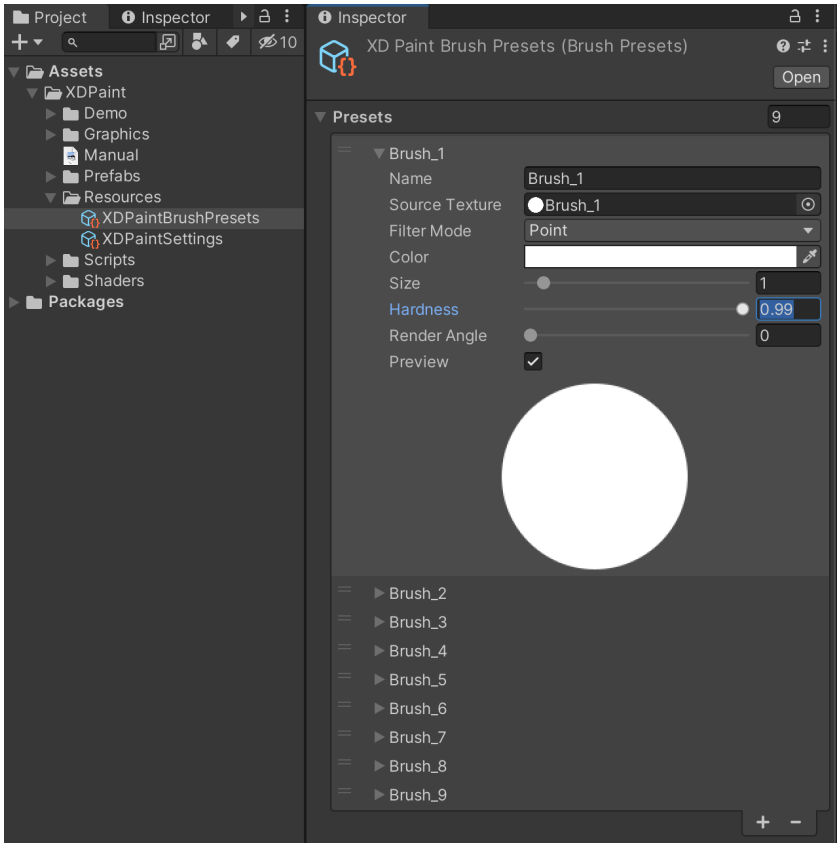
- **Brush** - default tool for painting;
- **Erase** - erase drawing result;
- **Eyedropper** - pick up a color and set it as a color of brush;
- **BrushSampler** - copying a part of drawing area to new texture;
- **Clone** - tool for cloning parts of the texture; Clone tool works the same way as in photoshop or any other painting software. First click on the object will take the start position of cloning, all follow-up clicks will clone parts of texture;
- **Blur** - tool for blurring parts of the texture. Blur implementation using simplified Gaussian blur algorithm that works in two Shader Passes and can be configured to change blur strength. Blur tool has such parameters:
  - **Iterations** - iterations count for blurring, higher value means strengthen blur; It is recommended to use parameter in range from 1 to 5. Higher value also means that for blur will be used more GPU resources and draw calls count will be increased (two draw calls per iteration);
  - **BlurStrength** - strength of blur. Zero means minimal blur, higher value - strengthen blur;
  - **DownscaleRatio** - value for quantity of times to downscale the texture size for blurring.
- **Gaussian Blur** - tool for blurring parts of the texture. Blur implementation using Gaussian blur algorithm that works in one Shader Pass and can be configured to change blur strength. Blur tool has such parameters:
  - **Kernel size** - iterations count for blurring, higher value means strengthen blur; It is recommended to use parameter in range from 3 to 7;
  - **Spread** - spreading of blur. Zero means minimal blur, higher value - strengthen blur. It is recommended to use parameter in range from 0 to 5;

Brushes and Presets

Asset has preset system to store brush parameters in ScriptableObject, that allows to reuse them with PaintManagers/PaintController. Presets located in the file at: Assets/XDPaint/Resources/XDPaintBrushPresets.asset. User can save brush parameters with name and apply it to other objects from the list. If user changes non-custom parameters of Brush in PaintManager/PaintController inspector window, data won't be written to Presets automatically.

Asset has built-in presets that can be reused or modified. Presets can be re-saved using PaintController/PaintManager Inspector window or modified by selecting XDPaintBrushPresets asset in Project window and editing in Inspector window.



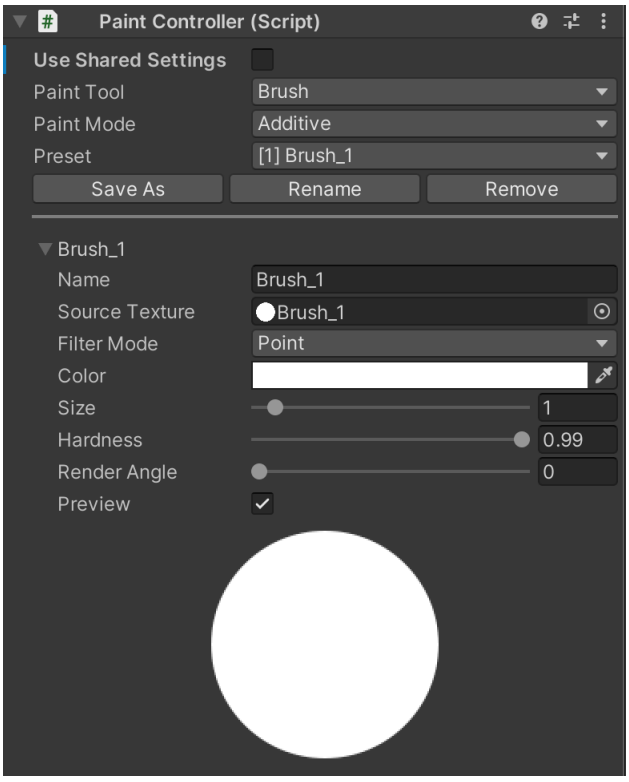


Preset can be re-saved to a new preset, for this click on "**Save As**" button.

Preset can be renamed, for this click on "**Rename**" button and choose a new name and click on "**Save**" button.

Preset can be removed, for this click on "**Remove**" button and confirm it using dialogue box.

User can use one brush parameters for all PaintManagers, for that check Use Shared Settings in PaintController, otherwise each PaintManager will have unique brush parameters.



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 three RenderTexture for each paint object - Paint Texture, Input Texture and Combined Texture.

The Paint Texture is using for painting, Input Texture stores user drawing result for current frame (in case when user use Default Paint Mode) or drawing result from mouse down event to mouse up event (in case when user use Additive Paint Mode). Input textures blends into PaintTexture each frame in Default Paint Mode and on mouse up event in Additive Paint Mode;

Combined Texture - texture that contains Source Texture as background (optional), Paint Texture as drawing texture, Input Texture as result of current drawing and preview of the brush (optional). Combined Texture - texture that user sees on the display when draws.

Depends on the selected tool, count of RenderTextures may vary.

Paint Modes

From version 2.4 asset has two paint modes: Default and Additive. Default mode bakes draw result into Paint Texture each frame, Additive mode provides more accurate color and alpha blending, bakes draw result in Paint Texture on Mouse Up event.

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 render 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

PaintController class

Script-singleton that stores all PaintManagers. Can set shared settings for all PaintManager when flag “Use Shared Settings” is checked, otherwise each PaintManager will have its own settings.

Main public fields, properties and methods:

public bool UseSharedSettings - whether to use settings of PaintController (Brush, Paint Mode, Tool and Preview) for all PaintManagers;
public PaintTool Tool { ... } - current tool;
public Brush Brush { ... } - returns Brush instance;
public void RegisterPaintManager(PaintManager paintManager) - register PaintManager;
public void UnRegisterPaintManager(PaintManager paintManager) - unregister PaintManager;
public IPaintMode GetPaintMode(PaintMode mode) - returns instance of paint mode;
public PaintManager[] ActivePaintManagers() - returns active PaintManagers;
public PaintManager[] AllPaintManagers() - returns all registered PaintManagers;

InputController class

Script-singleton is a component for user input management.

Main public fields, properties and methods:

public bool IsVRMode - whether to use VR mode instead of Mouse / Touch input;
public Transform HandTransform - Transform of Hand for painting;
public Canvas Canvas - canvas to ignore raycasts;
public GameObject[] IgnoreForRaycasts - GameObjects to ignore raycasts.
public event OnInputUpdate OnUpdate - on input update;
public event OnInputPosition OnMouseHover - mouse hover event on objects;
public event OnInputPositionPressure OnMouseDown - left mouse click event on objects;
public event OnInputPositionPressure OnMouseButton - left mouse button pressing event on objects;
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 class

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;

PaintManager class

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

Main events, fields, properties and methods:

public event InitHandler OnInitialized - initialization finished event, returns PaintManager instance;
public event DisposeHandler OnDisposed - PaintManager resources disposed event;
public GameObject ObjectForPainting - GameObject of the object to be painted;
public FilterMode FilterMode - property of the FilterMode of RenderTextures;
public Brush Brush - property of the brush of PaintManager;
public ToolsManager ToolsManager - property of the ToolsManager, contains and manages all the tools for painting;
public PaintTool Tool - property of the current paint tool;
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 DoDispose() - destroys all PaintManager created RenderTextures, Meshes and Materials, restore source material and texture;
public void Render() - invokes object rendering;
public void SetPaintMode(PaintMode paintMode) - sets paint mode;
public IPaintMode GetPaintMode() - returns paint mode instance;
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 GetRenderTextureLine() - returns RenderTexture of painting for current frame (used by clone and blur tools);
public RenderTexture GetPaintTexture() - returns RenderTexture of painting;
public RenderTexture GetPaintInputTexture() - returns RenderTexture of input (current frame dot/line for Default PaintMode or texture that was drawn between mouse down and mouse up events for Additive PaintMode);
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);
public void InitBrush() - initialize brush settings.

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 event DrawPointHandler OnDrawPointHandler - draw point event, can be used by the developer to obtain data about painting;
public event DrawLineHandler OnDrawLineHandler - draw line event, can be used by the developer to obtain data about painting;
public bool IsPainting { ... } - property, whether user is painting;

public bool IsPainted { ... } - property, whether user is painting (in current frame vertices were drawn);
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 DoDispose() - 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 Render() - renders to RenderTexture, in the case when using two RenderTexture's (PaintManager.renderTextureMode as RenderTexturesMode.TwoTextures);
public void ClearTexture(bool writeToUndo = false) - clears the texture to paint, to display changed data call paintManager.Render() method.

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 and events:

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;

public event Action<bool> OnUndoStatusChanged() - event of undo status changing, returns bool value if user can undo;
public event Action<bool> OnRedoStatusChanged() - event of redo status changing, returns bool value if user can redo;

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 DoDispose() - 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.

Brush class

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

Basic public properties and methods:

public string Name { ... } - brush name, must be unique;
public Material Material { ... } - brush material;
public FilterMode FilterMode { ... } - FilterMode of the RenderTexture of the brush;
public Color Color { ... } - brush color;
public Texture SourceTexture { ... } - source texture of the brush;
public RenderTexture RenderTexture { ... } - renders texture of the brush;
public Vector2 SourceTextureSize { ... } - size of the Source Texture;
public float MinSize { ... } - minimal size of the brush;
public float Size { ... } - brush size;
public float RenderAngle { ... } - brush render angle in degrees;
public Quaternion RenderQuaternion { ... } - brush render quaternion;
public float Hardness { ... } - brush hardness;
public bool Preview { ... } - brush preview;
public ChangeColorHandler OnChangeColor; - event of changing the brush color;
public ChangeTextureHandler OnChangeTexture; - event of changing the texture of the brush;
public void Init(IPaintMode mode) - initializing the brush Material, Mesh and RenderTexture;
public void DoDispose() - destroys previously created RenderTexture, Mesh and Material;
public void SetValues(Brush brush) - sets values for the brush from other brush;
public void Render() - render brush into RenderTexture;

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, bool canUpdateRenderTexture = true) - sets the brush texture;
public void SetPaintTool(PaintTool paintTool) - sets current tools for changing shader params;
public void SetPaintMode(IPaintMode mode) - sets paint mode.

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() - initializing for PaintManager, subscribe for PaintManager events;
public void SetTool(PaintTool paintTool) - sets tool;
public void DoDispose() - releases tools resources;

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 PaintManager PaintManager { ... } - PaintManager of the tool;
public virtual bool ShowPreview { ... } - whether to show preview or not;
public virtual bool DrawPreview { ... } - whether to draw preview into texture or not;
public virtual bool RenderToPaintTexture { ... } - whether to render to PaintTexture or not;
public virtual bool RenderToInputTexture { ... } - whether to render to InputTexture 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 DrawPreProcess { ... } - whether to draw pre process data or not;
public virtual bool DrawProcess { ... } - whether to draw process data or not;
public virtual bool BakeInputToPaint { ... } - whether to bake input texture to paint texture or not;
public virtual void Enter() - enter to the tool;
public virtual void Exit() - exit from the tool;
public virtual void UpdateHover(BasePaintObject sender, Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseHover;
public virtual void UpdateDown(BasePaintObject sender, Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseDown;
public virtual void UpdatePress(BasePaintObject sender, Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseButton;
public virtual void OnPaint(BasePaintObject sender, Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnPaintPointOrLine;

public virtual void UpdateUp(BasePaintObject sender, Vector2 uv, Vector2 paintPosition, float pressure) - invokes on BasePaintObject.OnMouseUp;
public virtual void OnDrawPreProcess(BasePaintObject sender, CommandBuffer commandBuffer, RenderTargetIdentifier rti, Material material) - invokes before combining textures in current frame;
public virtual void OnDrawProcess(BasePaintObject sender, CommandBuffer commandBuffer, RenderTargetIdentifier rti, Material material) - combining textures in current frame;
public virtual void OnBakeInputToPaint(BasePaintObject sender, CommandBuffer commandBuffer, RenderTargetIdentifier rti, Material material) - baking input texture into paint texture;
public virtual void OnUndo(BasePaintObject sender) - invokes on BasePaintObject.TextureKeeper.OnUndo;
public virtual void OnRedo(BasePaintObject sender) - invokes on BasePaintObject.TextureKeeper.OnRedo.

AverageColorCalculator class

Script for getting an average color of the texture.

Main public fields and methods:

public PaintManager PaintManager - paint manager for getting average color;
public PaintRenderTexture PaintRenderTexture - texture to check average color;
public bool SkipAlphaPixels - whether to skip source texture alpha pixels;
public ColorHandler OnGetAverageColor - event for getting average color. Invokes when the user paints;
public void SetAccuracy(int accuracy) - sets sampling accuracy.

AverageColorCalculator gets average color using shader «XD Paint/Average Color» that samples texture. It has two parameters: main texture and accuracy - divider for getting samples count. Smaller quantity means better accuracy and as a result - more GPU resources will be used. As an example, if texture has size 2048x1024 and accuracy has value 64, it will sample texture 2048 / 64 = 32 times per horizontal x 1024 / 64 = 16 times per vertical, in sum 32 x 16 = 512 samples. All this works on GPU which provides best performance.

Frequently used methods

Check if there is ability to undo/redo:

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

Another way is to subscribe to change state events:

```
PaintManager.PaintObject.TextureKeeper.OnUndoStatusChanged += OnUndoStatusChanged;
PaintManager.PaintObject.TextureKeeper.OnRedoStatusChanged += OnRedoStatusChanged;

private void OnUndoStatusChanged(bool canUndo)
{
    Debug.Log("Can undo: " + canUndo);
}

private void OnRedoStatusChanged(bool canRedo)
{
    Debug.Log("Can redo: " + canRedo);
}
```

Undo/Redo action:

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

Remove undo/redo data:

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

Repaint object:

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

Set default texture size for objects without source texture:

```
PaintManager.Material.DefaultTextureWidth = …;
PaintManager.Material.DefaultTextureHeight = …;
```

Clear render texture:

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

Brush size:

```
PaintManager.Brush.Size = value;
```

Brush hardness:

```
PaintManager.Brush.Hardness = value;
```

Change brush color:

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

Change brush opacity:

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

Change brush angle:

```
PaintManager.Brush.RenderAngle = value;
```

Change tool:

```
if (PaintController.Instance.UseSharedSettings)
{
  //In case when PaintController has checked UseSharedSettings
  PaintController.Instance.Tool = PaintTool..;
}
else
{
  //Otherwise:
  PaintManager.Tool = PaintTool...;
}
```

Change blur tool parameters:

```
var blurTool = PaintManager.ToolsManager.CurrentTool as BlurTool;
if (blurTool != null)
{
  blurTool.Iterations = 3;
  blurTool.BlurStrength = 2f;
  blurTool.DownscaleRatio = 1;
}
```

Change gaussian blur tool parameters:

```
var gaussianBlurTool = PaintManager.ToolsManager.CurrentTool as GaussianBlurTool;
if (gaussianBlurTool != null)
{
  gaussianBlurTool.KernelSize = 3;
  gaussianBlurTool.Spread = 5f;
}
```

Enable/Disable input for all PaintManagers:

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

Enable/Disable input for some PaintManager:

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

Getting average color of PaintManager:

```
public AverageColorCalculator AverageColorCalculator;
…
  AverageColorCalculator.OnGetAverageColor += color =>
      Debug.Log("Average Color: " + color);
```

Saving painting result to file:

```
var path = … ; //path for texture
var texture2D = PaintManager.GetResultTexture();
var pngData = texture2D.EncodeToPNG();
if (pngData != null)
{
  File.WriteAllBytes(path, pngData);
}
```

Drawing from code

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

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

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

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

```
// screen position in pixels
Vector3 screenPos = new Vector3(...);
var ray = Camera.main.ScreenPointToRay(screenPos);
RaycastController.Instance.Raycast(ray, out var 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:

```
// screen position in pixels
Vector3 screenPos = new Vector3(…);
```

then use functions for painting:

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

Drawing using collisions is possible, for that user need to get texture coord of hitting and calculate new pixel-coordinates for painting:

```
    private void OnCollisionEnter(Collision collision)
    {
      RaycastHit hit = new RaycastHit();
      Ray ray = new Ray(collision.contacts[0].point - collision.contacts[0].normal,
collision.contacts[0].normal);
      if (Physics.Raycast(ray, out hit))
      {
            Debug.Log(hit.textureCoord);
            var sourceTexture = PaintManager.Material.SourceTexture;
            var position = new Vector2(hit.textureCoord.x * sourceTexture.width,
hit.textureCoord.y * sourceTexture.height);
            PaintManager.PaintObject.DrawPoint(position);
      }
    }
```

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();
}
```

VR Support

«2D/3D Paint» supports work with VR. As user can used different VR plugins it is necessary to modify InputController for property work with input devices:

- Uncomment first line of InputController (`#define VR_ENABLED`);
- Configure VR input devices in `void Start()` method (current body of method can be replaced);
- Replace mouse input lines (`//next line need to be changed to VR device input`) to VR input, as example:

```
//next line need to be changed to VR device input
if (Input.GetMouseButtonDown(0))
```

Replace with:

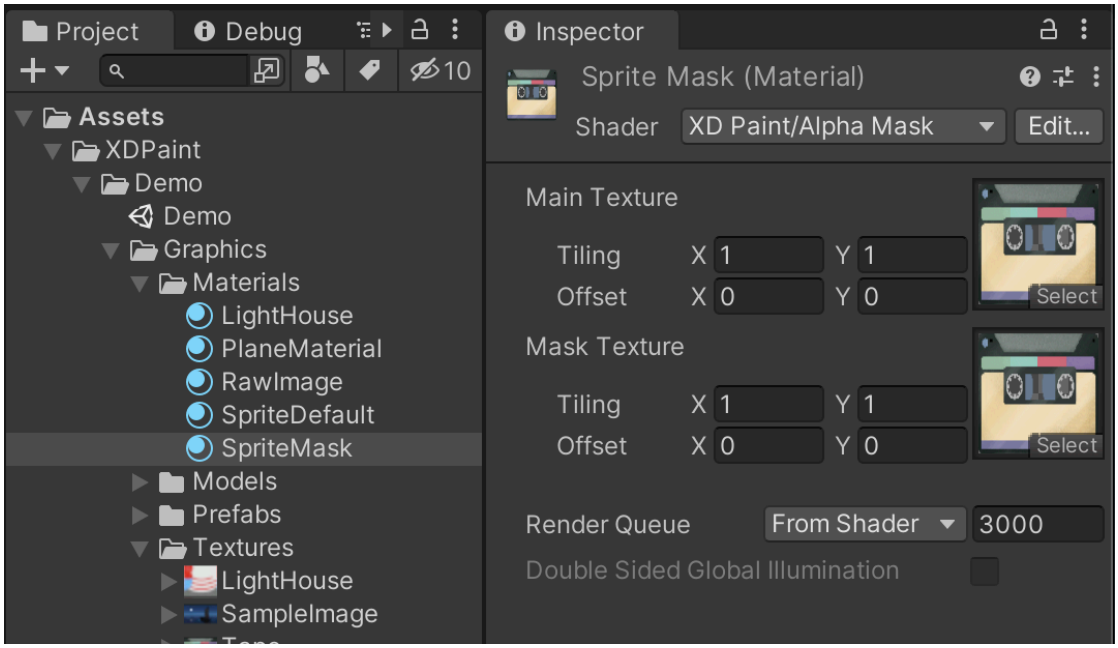
```
//next line need to be changed to VR device input
if (leftHandedControllers.Count > 0 &&
leftHandedControllers[0].TryGetFeatureValue(CommonUsages.triggerButton, out var
triggerValue) && triggerValue)
```

Set InputController field «Is VR Mode» as true and choose «Pen Transform» as your pen.
You can also use any other VR-plugins with built-in Input logic.
That's all, your VR-device is ready for painting!

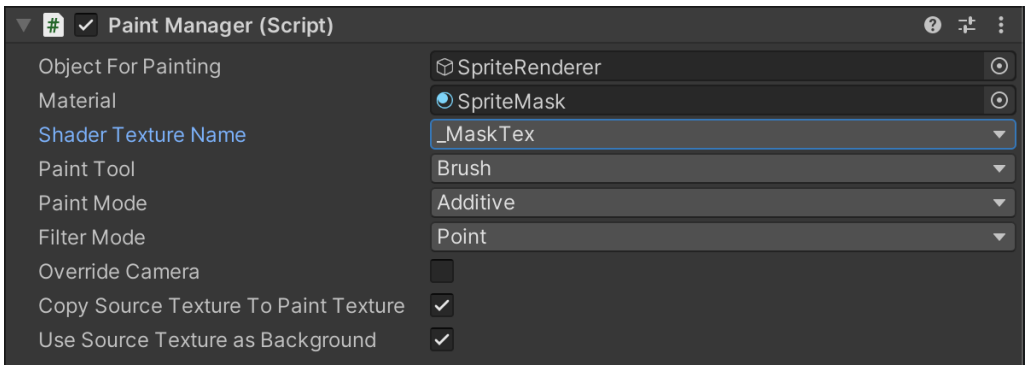
Tips

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

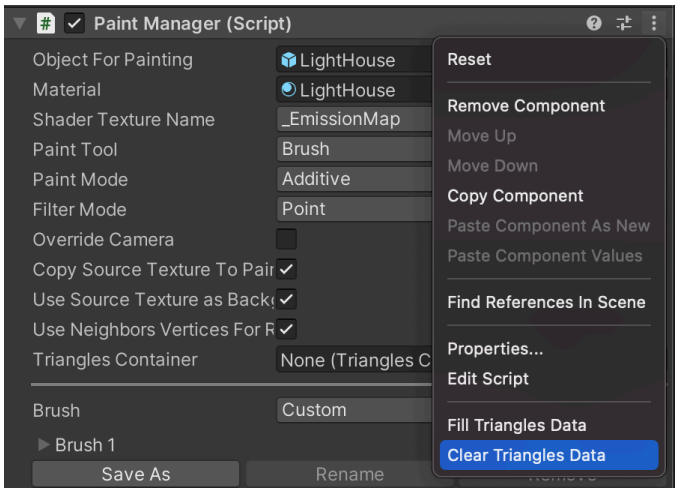
- Using a custom brush, make sure that their textures «Wrap Mode» are set as «Clamp» in order to avoid duplication of brushes with preview mode on;
- To disable the roundning of the brush, set the Brush.Hardness parameter to 1;
- Use the «Use Neighbors Vertices For Raycasts» for **PaintManager** if it 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;
- To paint on Sprite with alpha areas, use Material with Shader “XD Paint/Alpha Mask” and set Sprite as Main Texture and Mask Texture:

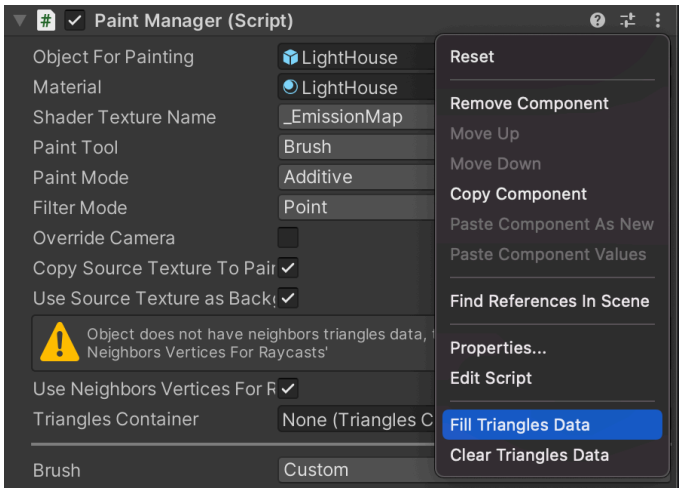


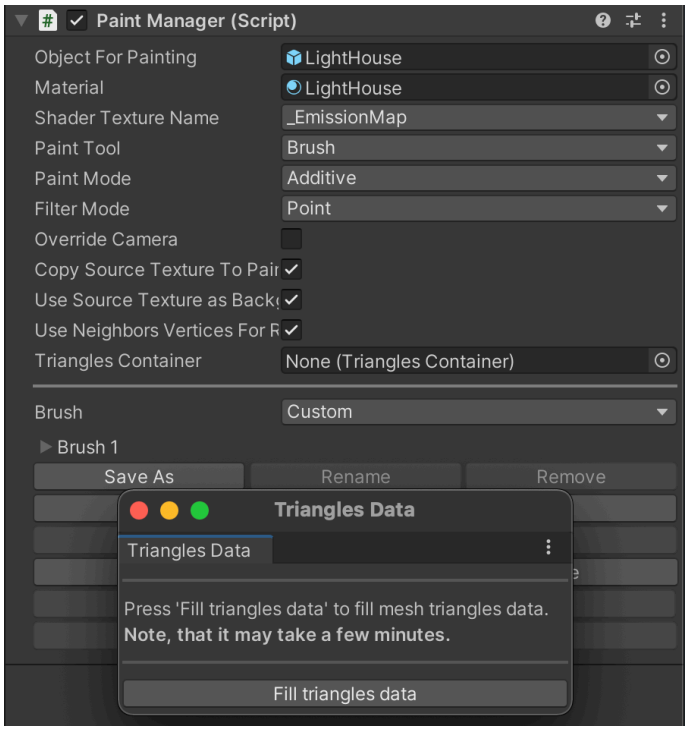
After that choose Shader Texture Name as “_MaskTex”:



- 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