Documentation **V15**

Mesh Deformation Full Collection

Last package overview

Last package video-documentation

It's recommended to open the documentation in the browser to keep up-to-date

Older package video-resources

Package Documentations

Full package-related playlist can be found here

Package Overviews

[Online resources don't have to be the same version, but they are related to the actual version]

Change Log

15th Update (01/03/2021 dd/mm/yyyy)

- Added MDM_MeshEffector modifier
- Added MD_EditorUtilities for modular editor
- Added MD_MeshProEditor modular utilities class
- Added MD_PathCreator system
- Added MD_Preferences editor window for global settings
- Added brand new examples called MESHLAB
- Upgraded overall VR support with modular setup
- VR support for all VR systems (XR, SteamVR, Oculus & Custom)
- All primitive generators refreshed & refactored
- MDM_Interactive 'Landscape' renamed to 'Surface'
- 'MDM_LandscapeTracking' renamed to 'MDM_SurfaceTracking'
- MD_SculptingPro - upgraded to v1.5
- MD_SculptingPro - removed additional window
- MD_SculptingPro - added history record
- MDM_FFD refactor & upgraded with new features
- MD_MeshColliderRefresher refactor
- MD_VertexTool & MD_VerticesSelector refactor
- MD_MeshPaint refactor
- MeshEditorRuntime refactor & massive upgrade
- MDM_MeltController refactor
- MDM_MeshDamage refactor
- MDM_MeshFit refactor
- MDM_MeshNoise refactor
- MDM_MeshSlime refactor
- MDM_RaycastEvent refactor
- MDM_Twist refactor
- MDM_Bend refactor
- MD_AdvancedShapes refactor
- MD_HexagonGrid refactor
- MD_TunnelCreator refactor
- Major editor icons & images upgrade

Content

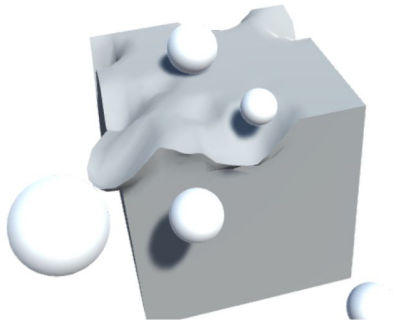
Basics

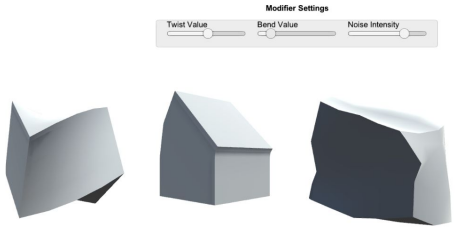
Basic information you should know

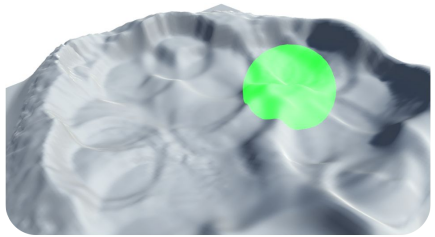
Introduction

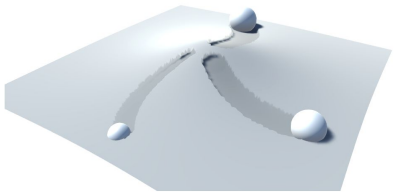
- **MD [Mesh Deformation]** is a collection of methods with mesh manipulation in Unity for beginners & advanced users. The package contains primitive vertex editor (PC,VR,Mobile), collection of various well-known modifiers, mesh collider refresher, shader-based deformation, physically based meshes, advanced shapes generator and many more. MD Package has many examples with details, source code, description & realtime support.
- The package began to be developed in 2014 by me, Matej Vanco. It started very simply with very basic features such as generating generic points for mesh, basic controls of Skinned Mesh bones and so on. The package was officially published on 12th of November 2015.
- Back in time the ‘mesh deformation’ was not very common in Unity, as the Unity is not a modelling software. This package contains many systems with unique & modular functionality. It's a collection of mesh deformations in Unity based on well-known modifiers & mesh features tested in many commercial projects by many users.

In this documentation, you will get familiar with all essential components, modifiers, shaders & additional tips and tricks about performance and overall use of the package.





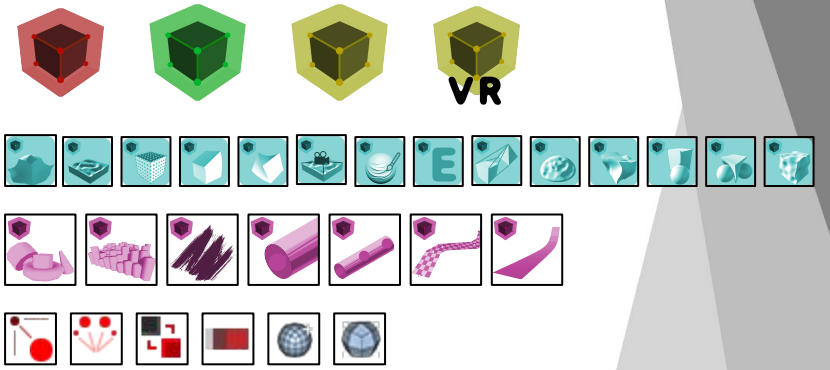




Summary

The **MD Package** is divided into **5 categories**:

- **Essentials**
  - *Contains essential components such as MeshProEditor, MeshEditorRuntime+VR & MeshColliderRefresher*
- **Modifiers**
  - *Contains all modifiers such as Mesh Damage, Sculpting, Interactive Surface, FFD and so on*
- **Shaders**
  - *Contains shader source called Standard Deformer*
- **Shapes**
  - *Contains all shape-generators such as Advanced Shapes, Tunnel Creator, Path Creator, Mesh Paint and so on*
- **Utilities**
  - *Contains additional utilities such as Mesh Smooth, Vertex Tools, Mesh Preferences and so on*



Each category has it's own folder with all required components. If you would like to use just modifiers, you can drag and drop the modifier onto any object with MeshFilter component. If you would like to generate procedural shape, you can create it in Hierarchy/Create. More about each category in next slides.

Essentials

First category with essential components

Full description & API

Editor-tooltips available

Global namespace = **MD_Plugin**

Mesh Pro Editor

Mesh Editor Runtime

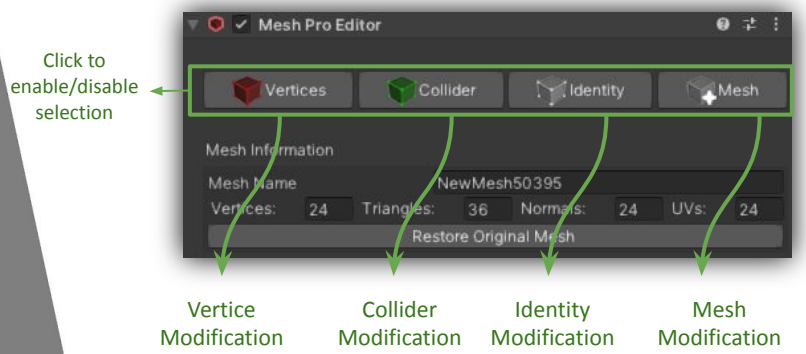
Mesh Editor Runtime VR

Mesh Collider Refresher

Editor Preferences

Mesh Pro Editor

Mesh Pro Editor is one of the essential components that allows you to analyze specific mesh, edit it's vertices & it features with a few external modifiers such as 'smooth mesh' & 'mesh subtraction'. The component is divided into **4 parts**: **Vertice Modification**, **Collider Modification**, **Identity Modification** & **Mesh Modification**. On the bottom of the component lies **Mesh Information** panel which contains essential data of the mesh (count of verts, tris etc). The component can be applied to objects that contain Mesh Filter or Skinned Mesh Renderer (It will be compiled to Mesh Filter either). It's highly recommended to use Mesh Pro Editor first before applying any other modifier. Mesh Pro Editor will ask you for 'creating a new mesh reference' and it's basically much safer while editing specific mesh. The component also contains rich API which can be used externally. The API is presented on the right side.



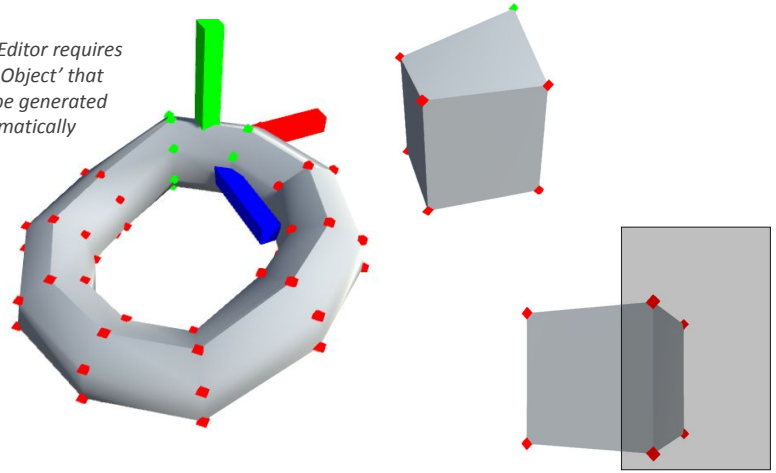
API - *Naming macro **MPE_***

public void MPE_ShowHideVertices(bool Activation)
[Show/ Hide points if possible - if their instance exists]
public void MPE_IgnoreRaycastVertices(bool IgnoreRaycast)
[Set 'Ignore Raycast' layer for generated points or vice-versa]
public void MPE_CreateVerticeEditor(bool PassTheVerticeLimit=false)
[Create vertice editor – points, if the parameter is true, the recommended vertice limitation will be ignored]
public void MPE_ClearVerticeEditor()
[Clear created vertice editor – points]
public void MPE_CombineMesh()
[Combine current mesh with it's submeshes (children). This will create brand new game object (safer way)]
public void MPE_CombineMeshQuick()
[Quick mesh combination with it's submeshes without any notification and new instance(Less safer way)]
public void MPE_CreateNewReference()
[Create new mesh reference & new game object, all your components on this object will be lost!]
public void MPE_RestoreMeshToOriginal()
[Restore current mesh to its original state]
public void MPE_ConvertFromSkinnedToFilter()
[Convert current Skinned Mesh to Mesh Filter (if possible)]
public void MPE_SmoothMesh(float Intensity)
[Smooth current mesh by specific value - internal built-in modifier]
public void MPE_SubdivideMesh(float Level)
[Subdivide current mesh by specific level - internal built-in modifier]

Mesh Editor Runtime

Mesh Editor Runtime allows you to edit generated points (by Mesh Pro Editor) at runtime by specified mouse input (PC) or “finger” input (Mobile). The component contains two types of runtime editor modes: **Axis Editor** & **Non Axis Editor**. The main difference is that the Axis Editor requires specific object with XYZ generics which will represent the well-known “axis arrow”, and Non Axis Editor allows you to drag & drop vertices. Both editor modes do the same job, but each has a different user-interaction and controls. The component is usually added to the main camera in scene and it’s specially built for designers without programming skills. It’s also possible to choose between three editor-control modes (Grab/Drop Vertex, Pull Vertex or Push Vertex).

Axis Editor requires ‘Axis Object’ that can be generated automatically



API - *Naming macro **NON_AXIS_ & AXIS_***

If AXIS MODE disabled

public void NON_AXIS_SwitchControlMode(int index)

[Change current control mode by index - Grab,Pull,Push]

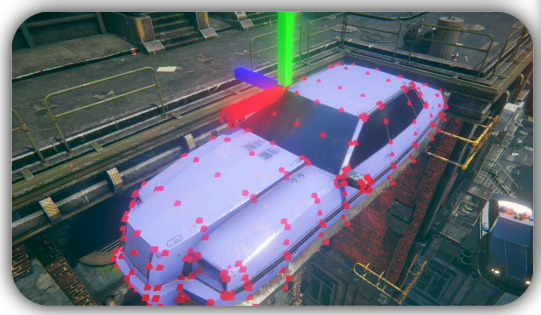
If AXIS MODE enabled

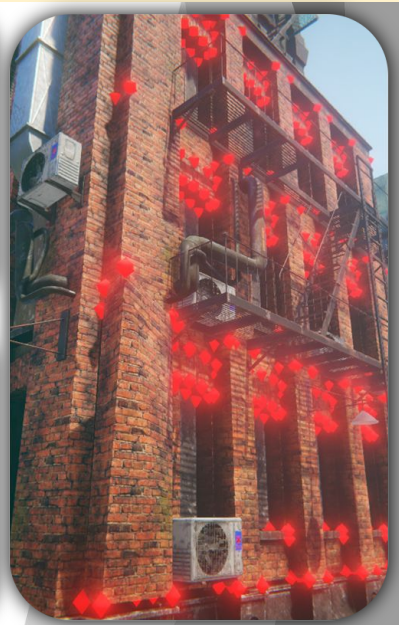
public void AXIS_SwitchTarget(MeshProEditor object)

[Change target for AXIS editor]

public void AXIS_Undo()

[Undo axis point selection]



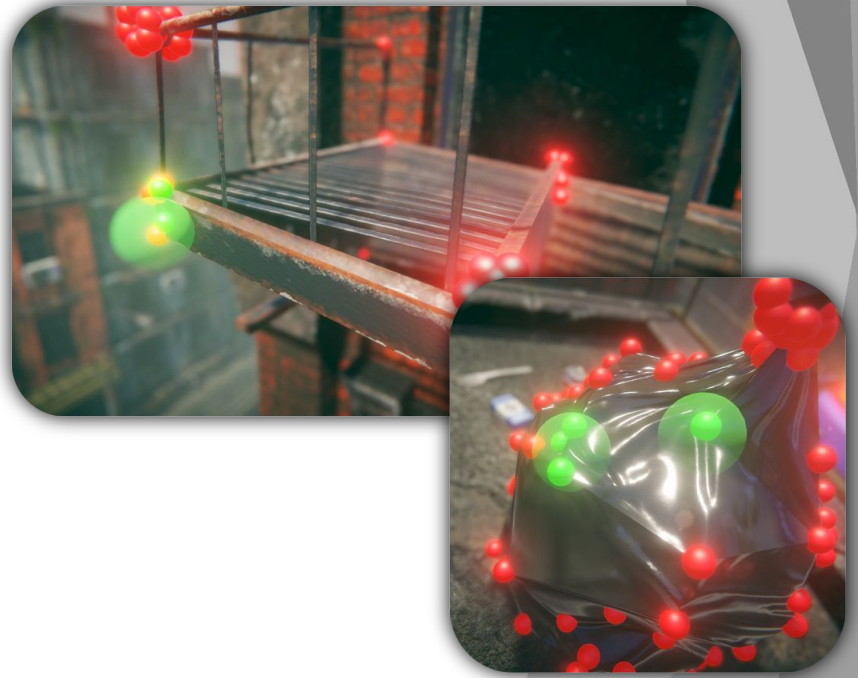


Mesh Editor Runtime VR

Mesh Editor Runtime VR allows you to edit generated points (by Mesh Pro Editor) at runtime by specified VR input. Just like the Non-VR Mesh Editor Runtime, it does the same job. All major VR platforms are fully supported including **SteamVR**, **Oculus Integration** and **Unity-XR**. It's required to choose & export proper package to prevent additional errors. It's also required to have properly installed **3rd package libraries** (e.g.: If you are going to use Mesh Editor Runtime VR for SteamVR, you should have installed SteamVR package from the Assets Store). The component should be added to one of the VR controllers. The overall VR setup for Mesh Editor Runtime VR is demonstrated in the VR Setup slide.

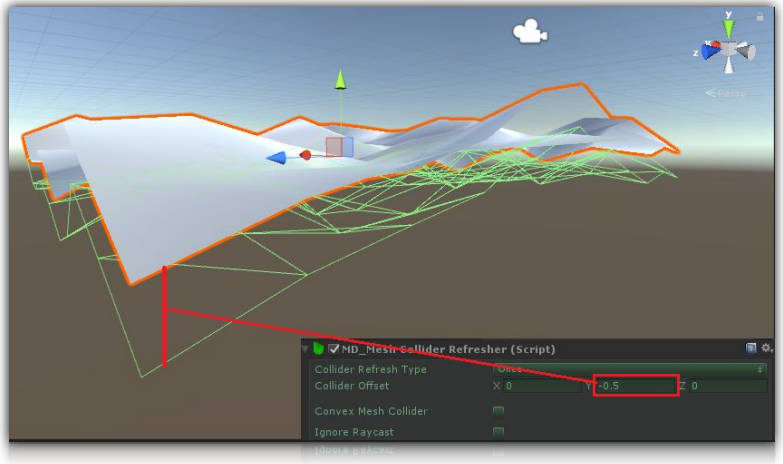
API - *Naming macro **VREditor_***

public void VREditor_SwitchControlMode(int index)
[Change current control mode by index - Move,Pull,Push]
public bool VREditor_GetControlInput()
[Get current built-in control VR input of the specified attributes. Returns true if pressed]
public void GlobalReceived_SetControlInput(bool setInputTo)
[Set control input from 3rd party source (such as SteamVR, Oculus or other)]



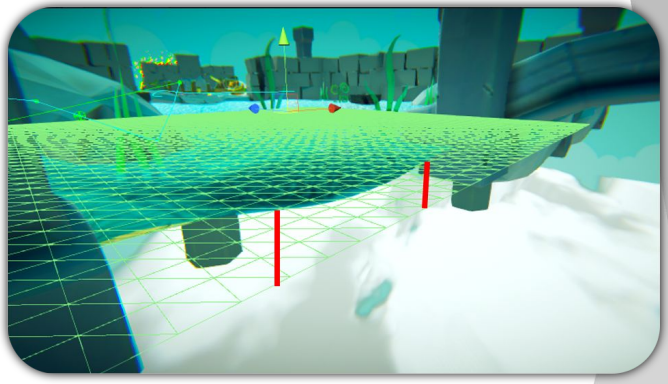
Mesh Collider Refresher

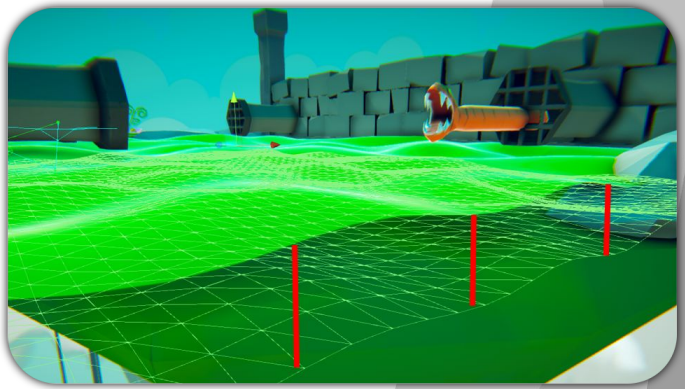
Mesh Collider Refresher allows you to update & refresh mesh collider at runtime. You can choose many modes such as refresh mesh collider once, every frame or by custom interval. You are also able to update the mesh collider manually or by the specific event action. In the refresh type 'once' you can set up your own mesh collider position offset. Simple example is described below.



API - *Naming macro **MeshCollider_***

public void MeshCollider_UpdateMeshCollider()
[Update mesh collider manually]

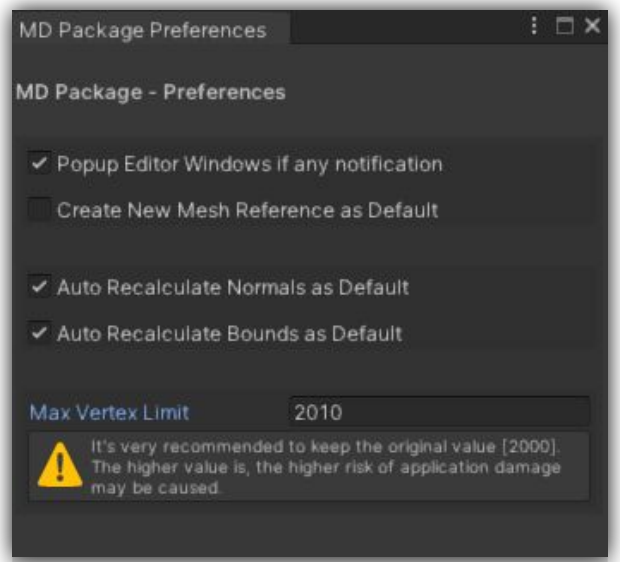




Preferences

The MD Package contains global preference window that helps you to edit specific tasks & features of the package. Preferences window can be found in **Window/MD_Package/Preferences**

The following image is an actual preferences window.



- **Popup editor window** if required with any component related to the MD Package (For example mesh is beyond the safe level of vertex count to edit)
- **Create new mesh reference** as default (Otherwise the original mesh data will remain = all modifiers create new references automatically)
- **Auto recalculate Normals & Bounds** as default (Otherwise the mesh will remain it's original normal & bounds data = all modifiers will recalculate bounds & normals automatically if enabled)
- Allowed **Max Vertex Limit** field. If mesh has more vertices than the specified value, the warning window will popup (if popup editor window is enabled for sure!)

Modifiers

Second category with modifiers

Full description & API

Editor-tooltips available

Global namespace = **MD_Plugin**

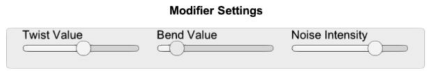
MDM_Bend
MDM_Twist
MDM_MeshNoise
MDM_FFD
MDM_MeshEffector
MDM_MeshDamage
MDM_Morpher
MDM_InteractiveSurface
MDM_SurfaceTracking
MDM_MeshFit
MDM_MeshSlime
MDM_MeltController
MDM_SculptingLite
MDM_RaycastEvent

MDM_Bend,Twist,Noise

MDM *(MeshDeformationModifier)* **Bend**, **Twist** & **Noise** are one of the basic modifiers in the package. Each basic modifier deforms meshes in a unique way: Bend bends the mesh in 3 axis, Twist twists the mesh in 3 axis and Noise populates mesh & creates 'dirty' surface which can be modified in two modes - vertical and overall deformation. Overall noise controls whole mesh in 3d space while vertical noise controls mesh in 2d space - the mesh will be planar. All modifiers runs in editor & at runtime. All parameters can be changed at runtime as well.

All modifiers can be added to the object just once. Also it's **prohibited** to use **multiple modifiers** at **once** on the **same object** due to the vertices sync.

Bend, Twist & Noise are not multithreaded modifiers, so applying to hi-poly objects (2000 vertex count and more) is not recommended!



API - *Naming macro **Bend_, Twist_ & MeshNoise_***

Bend

public void Bend_ProcessBend()
[Process the bend effect itself]
public void Bend_RegisterCurrentState()
[Refresh & register current mesh state. This will refresh original vertices to the current state]
public void Bend_BendObject(UI Slider entry)
[Bend object by the UI Slider value]
public void Bend_BendObject(float entry)
[Bend object by the float value]

Twist

public void Twist_ProcessTwist()
[Process the twist effect itself]
public void Twist_RegisterCurrentState()
[Refresh & register current mesh state. This will refresh original vertices to the current state]
public void Twist_TwistObject(UI Slider entry)
[Twist object by the UI Slider value]
public void Twist_TwistObject(float entry)
[Twist object by the float value]

Noise

public void MeshNoise_UpdateVerticalNoise()
[Process vertical noise]
public void MeshNoise_UpdateGeneralNoise()
[Process general noise]
public void MeshNoise_ChangeIntensity(UI Slider entry)
[Change overall noise intensity by the UI Slider]
public void MeshNoise_ChangeIntensity(float entry)
[Change overall noise intensity by the float]

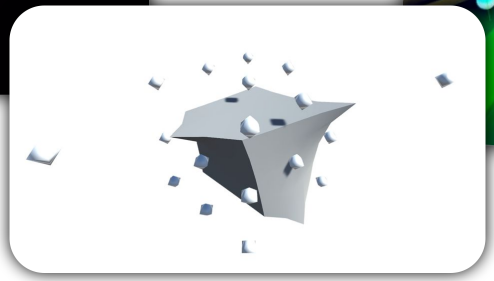
MDM_FFD

MDM *(MeshDeformationModifier)* **FFD** stands for Free-Form-Deformation which controls mesh by the registered weights in lattice. The mesh will smoothly deform by moving points in lattice. It's a great modifier to quickly edit desired mesh parts. The FFD modifier contains various lattice resolutions in cubic-shape: 2x2x2, 3x3x3, 4x4x4 and custom number (not recommended due to the performance).

FFD is not multithreaded modifier [yet], so editing hi-poly objects (2000 vertex count and more) is not recommended!

API - *Naming macro **FFD_***

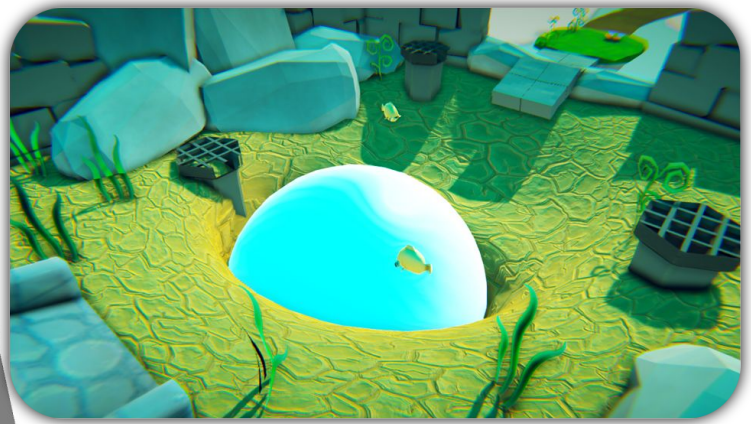
public void FFD_UpdateMesh()
[Update current mesh by FFD modifier]
public void FFD_RegisterWeights()
[Register current weights and 'bake' mesh]
public void FFD_RefreshFFDGrid()
[Refresh selected FFD type & its grid]
public void FFD_ClearFFDGrid()
[Clear FFD grid (if possible)]



MDM_MeshEffector

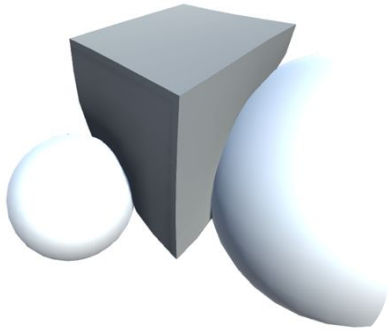
MDM *(MeshDeformationModifier)* **Mesh Effector** deforms mesh by registered weight nodes & density. There are four types of effectors: One-Pointed, Two-Pointed, Three-Pointed and Four-Pointed. Effectors create spherical radiuses in assigned nodes that affect the target mesh. In comparison with FFD, the Mesh Effector doesn't create lattice, instead it creates radiuses which deform overall mesh by several parameters such as Weight value, Weight multiplier, Weight density and Weight effector for each 'effector-type'. It's a great modifier for quick deformations without any further hard work!

Mesh Effector is multithreaded modifier, so it's safe to edit hi-poly objects (2000 vertex count and more).



API *- Naming macro **Effector_***

public void Effector_UpdateMesh()
[Update current mesh state (in case if Update Every Frame is disabled)]
public void Effector_ApplyWeights()
[Apply & register effector weights]



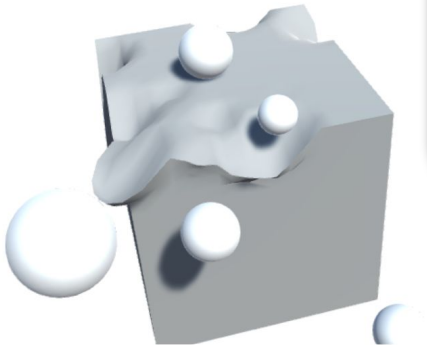
MDM_MeshDamage

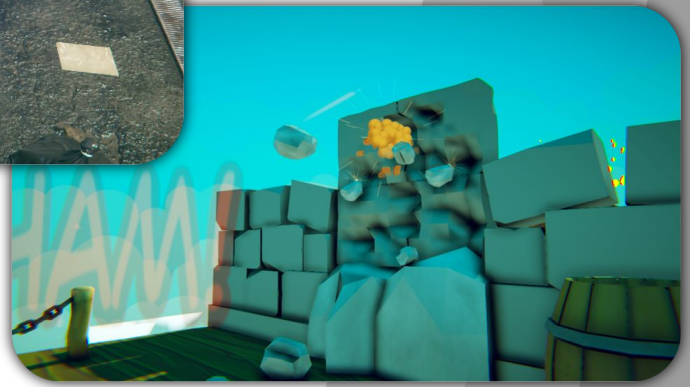
MDM *(MeshDeformationModifier)* **Mesh Damage** allows you to deform mesh surface with any external source such as rigidbody collision impact, raycast event and so on. Simple & very effective modifier that makes your mesh behaves more natural.

Mesh Damage is not multithreaded modifier, but it's safe to apply it on hi-poly objects (2000 vertex count and more) as the modifier is not heavy-to-calculate.

API - *Naming macro **MeshDamage_***

public void MeshDamage_ModifyMesh(Vector3 atPoint, float radius, float force, Vector3 initialDirection)
[Modify current mesh manually by the point, radius and force]
public void MeshDamage_RefreshVertices()
[Refresh vertices & register brand new original vertices state]
public void MeshDamage_RepairMesh(float speed)
[Repair deformed mesh by specified speed value]
public void MeshDamage_ModifyMesh(MDM_RaycastEvent RayEvent)
[Modify current mesh by custom RaycastEvent]





MDM_Morpher

MDM *(MeshDeformationModifier)* **Mesh Morpher** allows you to blend between various shapes. It's possible to choose unlimited shape count and register their initial vertex state. Morpher is well-known in face expressions, mimics, statue translations and so on.

Mesh Morpher is multithreaded modifier, so it's safe to edit hi-poly objects (2000 vertex count and more)

API - *Naming macro **Morpher_***

public void Morpher_ChangeMeshIndex(int entry)
[Change current target morph mesh index]
public void Morpher_SetBlendValue(Slider entry)
[Set current blend value]
public void Morpher_SetBlendValue(float entry)
[Set current blend value]
public void Morpher_RefreshTargetMeshes()
[Refresh target meshes - target meshes must be registered]
public void Morpher_UpdateMorpher()
[Update & refresh current morpher manually]

Example: Blend between 3 meshes

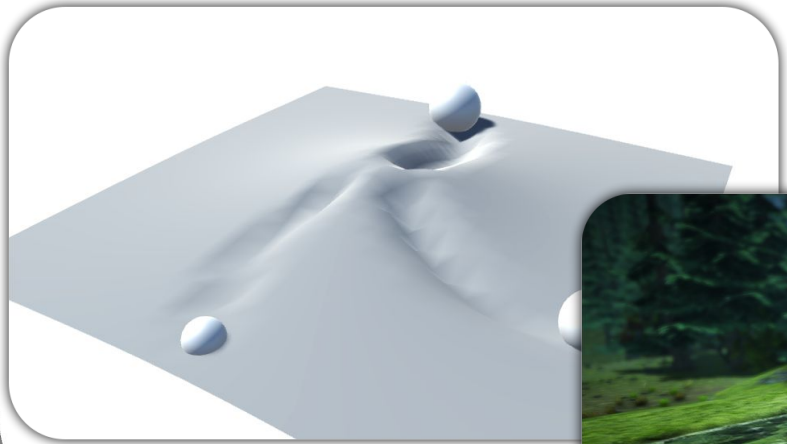
MDM_InteractiveSurface

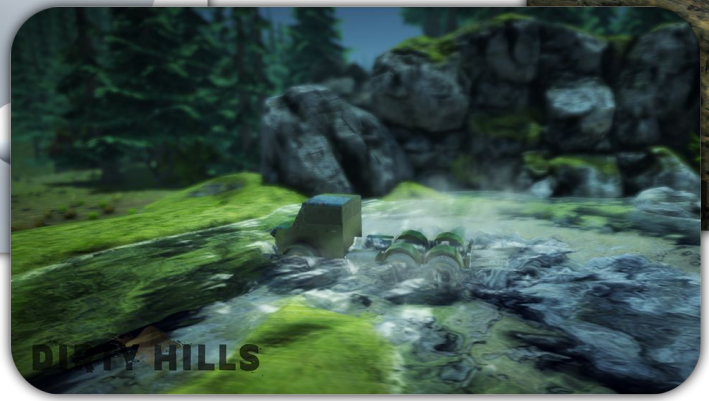
MDM *(MeshDeformationModifier)* **Interactive Surface** allows you to interact with any meshes and simulate ‘surface deformation’. Create dynamic tracks, snow trails, drops and more. The mesh can be ‘regenerated’ as well by specified speed value and interpolation.

Interactive Surface is multithreaded modifier, so it’s safe to edit hi-poly objects (2000 vertex count and more).

API - *Naming macro **InteractiveSurface_***

public void InteractiveSurface_ModifyMesh(Vector3 AtPoint, float Radius, Vector3 Direction)
[Modify mesh surface with specific point, size and vertice direction]
public void InteractiveSurface_ResetSurface()
[Reset current surface (Reset all vertices to the starting position)]
public void InteractiveSurface_ModifyMesh(MDM_RaycastEvent RayEvent)
[Modify current mesh by custom RaycastEvent]





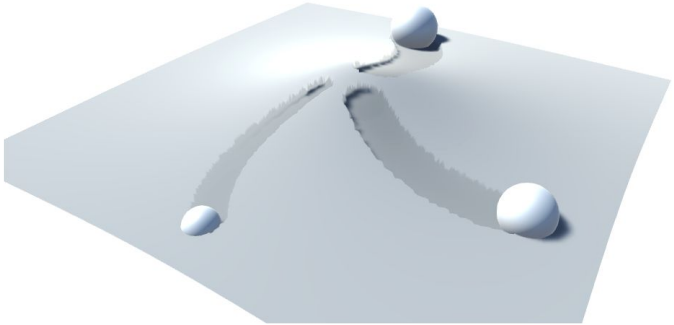
MDM_SurfaceTracking

MDM *(MeshDeformationModifier)* **Surface Tracking** does the same job as Interactive Surface, but requires extra material and extra settings to setup. Surface Tracking runs mostly via **GPU** which saves more performance and it's easier to play around. The modifier requires **MD_EasyMeshTracker** shader and Render Texture, which can be generated automatically in Inspector. If you would like to use the Surface Tracking for mobile, use **MD_EasyMeshTracker_Mobile** shader. Great modifier for quick surface simulations. In comparison with Interactive surface - the results are less smooth, depends on Tessellation value.

Surface Tracking is not multithreaded modifier, but it's safe to apply it on hi-poly objects (2000 vertex count and more) as the modifier is not heavy-to-calculate.

API - *Naming macro **SurfTracking_***

public void SurfTracking_ResetSurface()
[Reset current surface]



MDM_MeshFit

MDM *(MeshDeformationModifier)* **Mesh Fit** allows you to fit mesh to any surface with collider by generated or specified points. The modifier simply helps you to create universal decals and dynamic meshes with raycast events.

Mesh Fit is not multithreaded modifier, so editing hi-poly objects (2000 vertex count and more) is not recommended!

API - *Naming macro **MeshFit_***

public void MeshFit_ShowHidePoints(bool activation)
[Show/Hide generated points]
public void MeshFit_GeneratePoints()
[Generate points on mesh]
public void MeshFit_RestoreOriginal()
[Restore current mesh to its original state]
public void MeshFit_ClearPoints()
[Clear generated points (if possible)]
public void MeshFit_BakeMesh()
[Reset mesh matrix transform (Set scale to 1 and keep the shape)]
public void MeshFit_UpdateMeshState()
[Update mesh state manually]

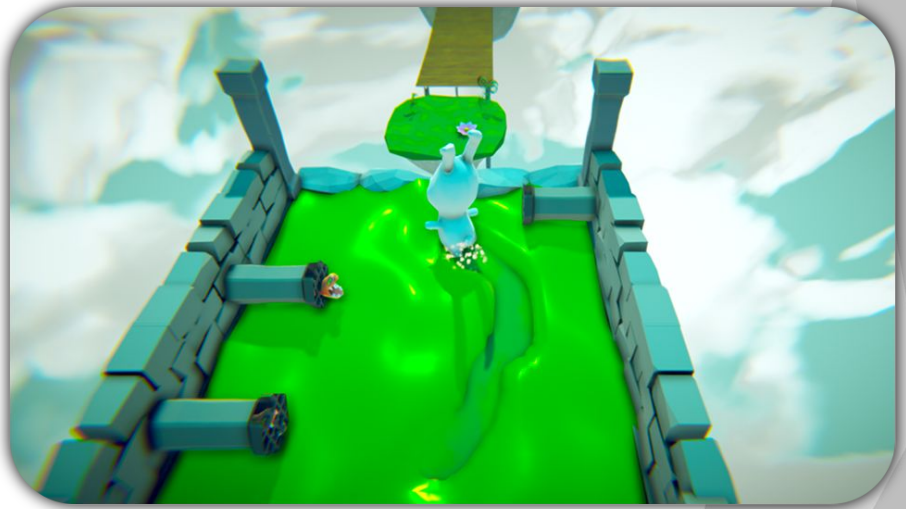
MDM_MeshSlime

MDM *(MeshDeformationModifier)* **Mesh Slime** allows you to create ‘slime-like’ surfaces with additional input. It basically does the same job as Interactive Surface, but with more settings & in more deeper job.

Mesh Slime is not multithreaded modifier, so editing hi-poly objects (2000 vertex count and more) is not recommended!

API - *Naming macro **SurfTracking_***

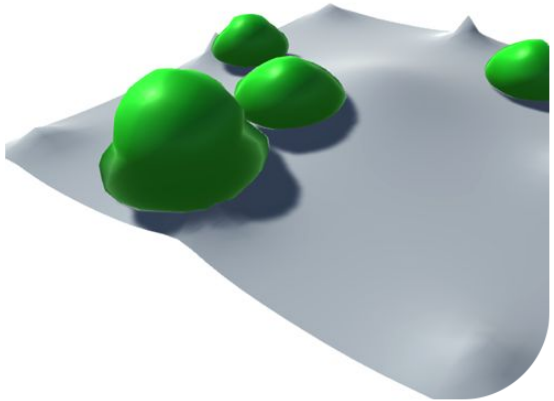
public void MeshSlime_ModifyMesh(Vector3 worldPoint)
[Modify mesh on specified world points]
public void MeshSlime_ModifyMesh(MDM_RaycastEvent entry)
[Modify mesh on specified Raycast event]



MDM_MeltController

MDM *(MeshDeformationModifier)* **Melt Controller** allows you to create simple melting effect. The controller requires special material with **Melting shader** which is included in the package. Setup melting zone, melt transition, melt amplification and more. This modifier does not work with HDRP or URP.

No API



MDM_SculptingLite

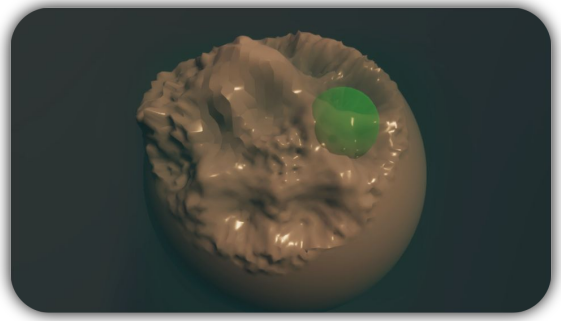
MDM *(MeshDeformationModifier)* **Sculpting Lite** allows you to deform any mesh in different & more-artistic way. The system opens a new perspective of mesh manipulation by radial-brush with various features. Brush type and brush appearance is fully customizable with brush radius & strength. It's also possible to use internal API which allows you to modify your own methods. The system is great for advanced terrain editing (Not built-in unity terrain) with various brush types like **smooth** (HCFilter, Laplacian Filter), **noise** filter, **stylization** and more. Sculpting pro is compatible for Mobile and VR as well and it's fully ready for extremely complex meshes thanks to multithreading.
The development of this mesh-tool has ended. Please visit brand new plugin called Sculpting Pro - the complete sculpting solution in Unity.

Sculpting Lite is multithreaded modifier, so it's safe to edit hi-poly objects (2000 vertex count and more). Tested on even much detailed meshes (50k vertex count and more).

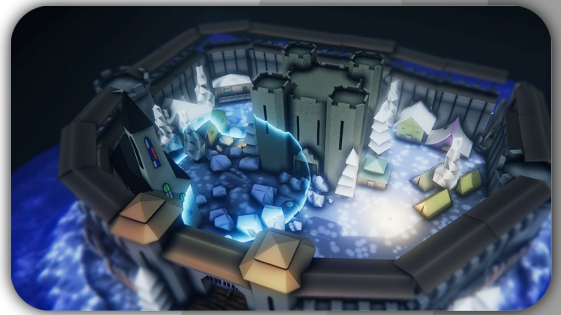
API - *Naming macro **SS_Funct_***

public void SS_Funct_RestoreOriginal()
[Restore mesh to the first original mesh]
public void SS_Funct_BakeMesh()
[Restart mesh transform and matrix]
public void SS_Funct_DoSculpting(Vector3, Vector3, float, float, enum)
[Process sculpting on the current mesh]
public void SS_Funct_RefreshMeshCollider()
[Refresh current mesh collider]
public void SS_Funct_ChangeBrushState(int)
[Change brush state 0 – none, 1 – raise, 2 – lower, 3 – revert, 4 - noise, 5 - stylize, 6 - smooth]
public void SS_Funct_SetBasics(float, float, bool, Vector3, Vector3)
[Set basic parameters for current sculpting object]
public void SS_Funct_ChangeRadius(Slider UI or float)
[Change radius of sculpt brush]
public void SS_Funct_ChangeStrength (Slider UI or float)
[Change strength of sculpt brush]
public void SS_Funct_RecordToHistory()
[Record current vertex position to the history (if possible)]
public void SS_Funct_Undo()
[Step back undo (if possible)]

Oculus Quest 2 with hand tracking



Used in PC & VR game The God



MDM_RaycastEvent

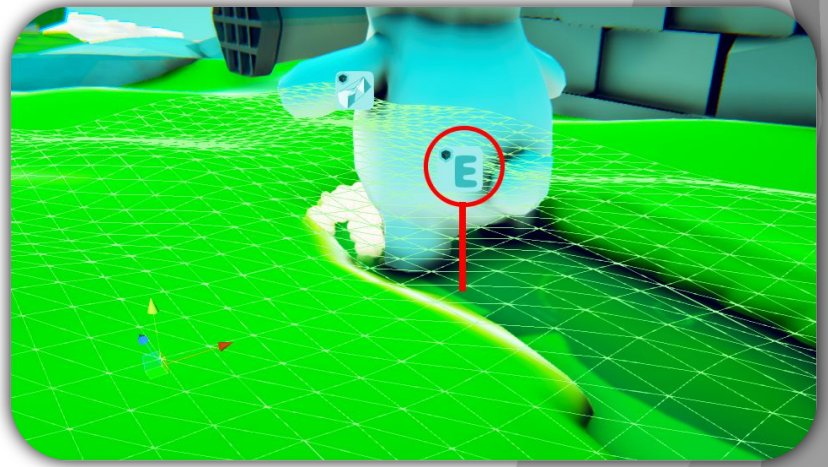
MDM *(MeshDeformationModifier)* **Raycast Event** is a simple modifier to render specified events if raycast hits something. You can set up your own event system & raycast logic without any programming skills. The raycast event is connected with almost all modifiers, so you can access their methods with Raycast Event.

API - *Naming macro **RayEvent_***

public void RayEvent_IsRaycasting()
[Is current raycast event raycasting?]
public void RayEvent_UpdateRaycastState()
[Check if raycast hits something manually]

Used as 'step-tracker' for Surface Tracking

Blabibo tracks its position for Mesh Slime



Shaders

Third category with shader source

Full description & API

Standard Deformer

Easy Mesh Tracker

Melt Shader

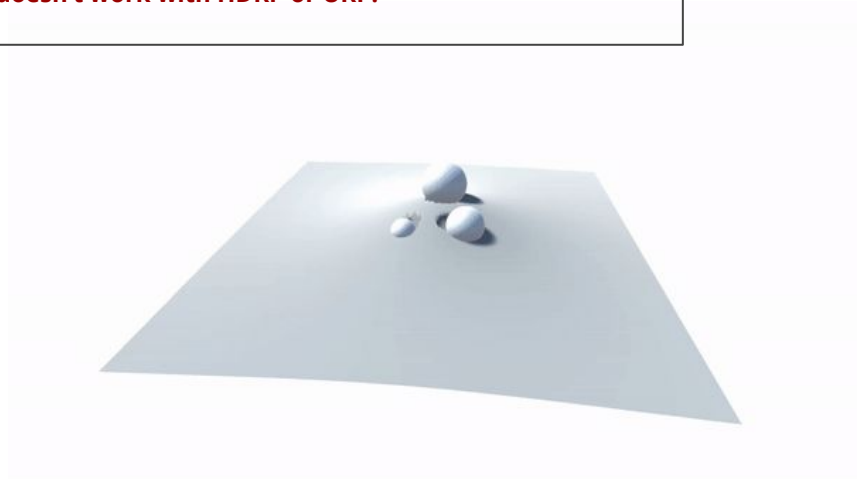
The following slides contain gif images. Please open the docs in browser, gif images don't work in PDF.

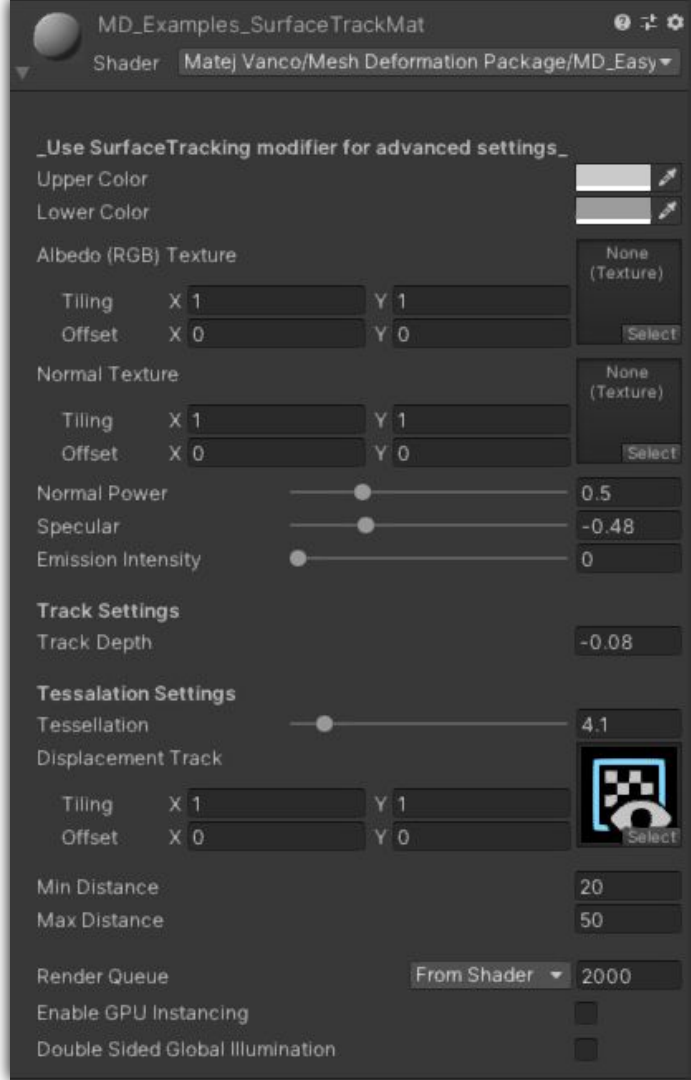
Standard Deformer

Standard Deformer is an essential shader in MD Package. The shader allows you to simply animate any object with renderer component in various ways. Create motion with many possibilities such as jumping, fish swimming, simple sway to all directions, noise simulation and so on. There is a game called Sea Orchestra which was made just from the Standard Deformer. Check the right panel with all available settings and features of the Standard Deformer. **The Standard Deformer doesn't work with HDRP or URP.**

Easy Mesh Tracker

Easy Mesh Tracker is a required shader for using **Surface Tracking** modifier. The modifier works with Render Textures which tell the shader how high or how low the mesh should be deformed. It's cheap and quick, so it's easy to set up! For using the Easy Mesh Tracker for mobile, use **Easy Mesh Tracker_Mobile**. Also the mobile version doesn't support Tessellation - you can use Procedural Plane to set custom vertice count. **The Easy Mesh Tracker doesn't work with HDRP or URP.**

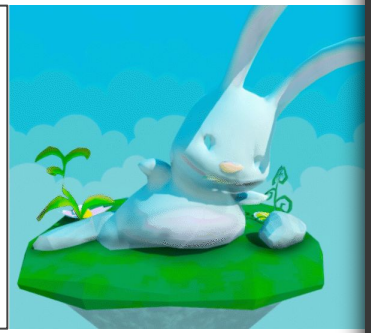


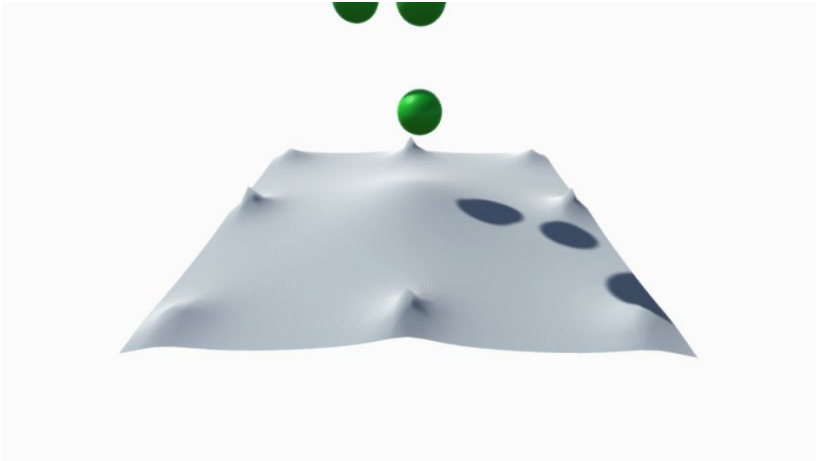


Melt Shader

Melt Shader is a required shader for using **Melt Controller**, however the shader can be used outside the Melt Controller modifier. The Melt Shader allows you to create ‘melting’ effect on any renderer object with various settings. Melt shader is compatible with all platforms.

The Melt Shader doesn’t work with HDRP or URP.





Shapes

Fourth category with shapes & generators

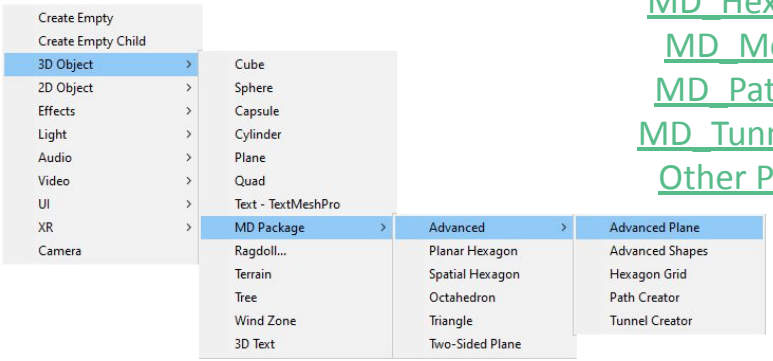
Full description & API

Editor-tooltips available

Global namespace = **MD_Plugin**

[MD_AdvancedPlane](#)
[MD_AdvancedShapes](#)
[MD_HexagonGrid](#)
[MD_MeshPaint](#)
[MD_PathCreator](#)
[MD_TunnelCreator](#)
[Other Primitives](#)

All shapes can be found in Hierarchy/Create



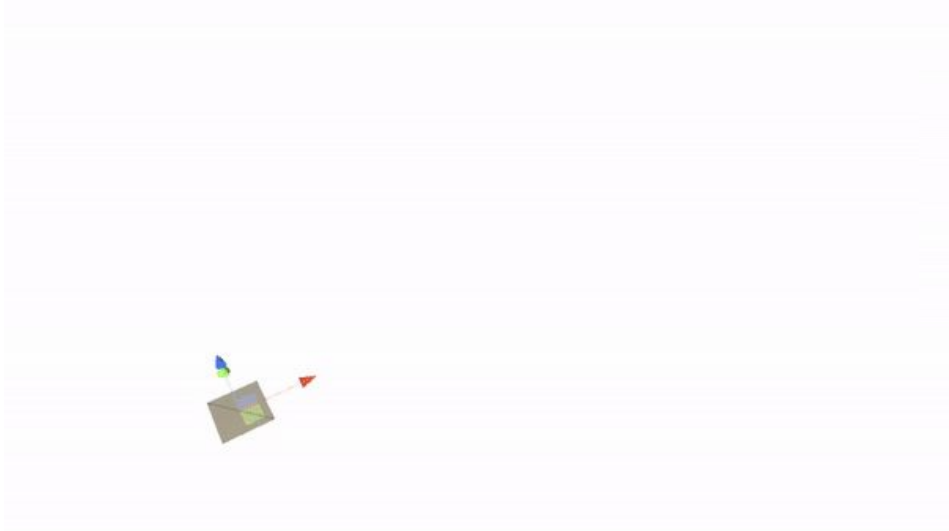
The following slides contain gif images. Please open the docs in browser, gif images don't work in PDF.

MD_AdvancedPlane

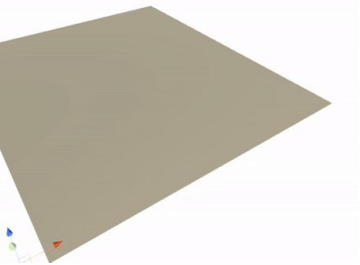
Advanced Plane is an advanced-shape component that allows you to generate procedural plane with custom vertex count. The main feature of the plane is that the generated plane has no smoothing, so the final results would be ‘sharp-looking’. The Advanced Plane contains an additional features called ‘Angle Property’ which allows you to ‘bend’ the plane up or down.

API - *Naming macro **AdvancedPlane__***

public void AdvancedPlane_Modify()
[Modify advanced plane manually (if the update feature is disabled)]
public static GameObject Generate()
[Generate custom advanced plane, returns gameobject with advanced plane component]



Noise modifier applied to the plane

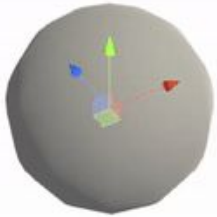


MD_AdvancedShapes

Advanced Shapes is a collection of procedural shapes with customizable vertex count. The component contains exactly 6 shapes (Plane, Box, Cone, Torus, Sphere, Tube), each shape has its own settings (width, height, length, vertex count - stack & segments).

API - *Naming macro **AdvancedShapes__***

public void AdvancedShapes_SwitchShape(int shapeID)
[Switch shape by index]
public static GameObject Generate()
[Generate custom advanced shape, returns gameobject with advanced shapes component]
public void AdvancedShapes_SwitchShape(ShapeType shape)
[Generate specified shape manually]



MD_HexagonGrid

Hexagon Grid is an advanced-shape component that allows you to generate procedural hexagon grid. Fully customizable hexagon grid with random height feature, that allows you to randomize each hexagon segments height. It's also possible to flip hexagon faces, make it planar or even change the size.

API - *Naming macro **HexagonGrid__***

public void HexagonGrid_ModifyHexagon()
[Modify hexagon manually]
public static GameObject Generate()
[Generate custom hexagon, returns gameobject with hexagon component]
public void HexagonGrid_ModifyPlanar()
[Modify planar hexagon manually]
public void HexagonGrid_ModifySpatial(float addHeightRand = 0)
[Modify spatial hexagon with custom height offset]
public void HexagonGrid_RandomizeHeight(float offset)
[Randomize hexagon height]

MD_MeshPaint

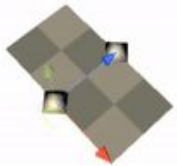
Mesh Paint is an advanced-shape component that allows you to paint mesh in three types: Plane, Triangle & Cube. The component is fully ready for designers with advanced settings of planar drawing or spatial drawing. The component is available for all platforms including VR & mobile.

API - *Naming macro **PUBLIC__***

public void PUBLIC_CreateNewPaintPattern(string MeshName = "TargetMesh", bool addCollider = true)
[Create new paint pattern mesh with custom attributes]
public void PUBLIC_PaintMesh(Vector3 Position, MeshPaintModeInternal MeshPaintMode)
[Paint mesh on the specific location by the selected method]
public void PUBLIC_ChangeBrushSize(float size)
[Change & set brush size manually]
public void PUBLIC_IncreaseBrushSize(float size)
[Increase brush size manually]
public void PUBLIC_DecreaseBrushSize(float size)
[Decrease brush size manually]
public void PUBLIC_ChangeBrushSize(UnityEngine.UI.Slider size)
[Change brush size manually by UI Slider]
public void PUBLIC_EnableDisableDrawing(bool activation)
[Enable/ Disable drawing externally]
public void PUBLIC_ChangeAppearanceIndex(int index)
[Change currently selected material/color by index]
public void PUBLIC_ChangeShapeType(int ShapeT)
[Change shape type (0 = Plane, 1 = Triangle, 2 = Cube)]
public void GlobalReceived_SetControlInput(bool setInputTo)
[Set control input from 3rd party source (such as SteamVR, Oculus or other)]

MD_PathCreator

Path Creator is an advanced-shape component that allows you to create & edit procedural path with triplanar mapping. Create simple paths with user-friendly nodes, edit tracks and apply further modifiers for smoothing features.

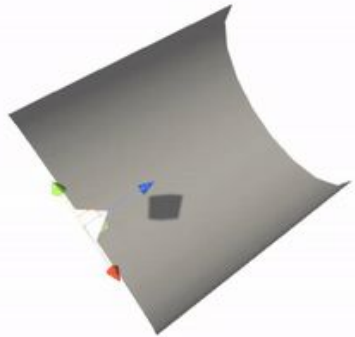


API - *Naming macro **PUBLICpFunct__***

public void PUBLICpFunct_AddNode(Vector3 toPos, bool GroupOnAdd = true)
[Add node on specific position]
public void PUBLICpFunct_RemoveNode()
[Remove last node]
public void PUBLICpFunct_ClearAll()
[Clear all nodes]
public void PUBLICpFunct_RefreshNodes()
[Refresh current tunnel mesh]
public void PUBLICpFunct_GroupNodesTogether()
[Group all nodes together in hierarchy]
public void PUBLICpFunct_UngroupNodes(Transform Detachto)
[Ungroup all nodes to 'empty' or to 'some object']
public void PUBLICpFunct_UpdateUVs()
[Update UV sets with specific UV mode]
public static void GeneratePathObj()
[Create brand new gameObject with path creator]

MD_TunnelCreator

Tunnel Creator is an advanced-shape component that allows you to create & edit procedural tunnel with various texture mapping, including custom. All the texture mappings are based on triplanar mapping. The system is based on nodes and weights. There also many additional options to make a turn, straight line or connect other ‘tunnels’. Tunnel Creator also contains its own editor window to make it more user friendly. Tunnel Creator system contains 2 important components: **MD_TunnelCreator** (the root) and **MD_TunnelNodeUVData** (the node uv data controller).



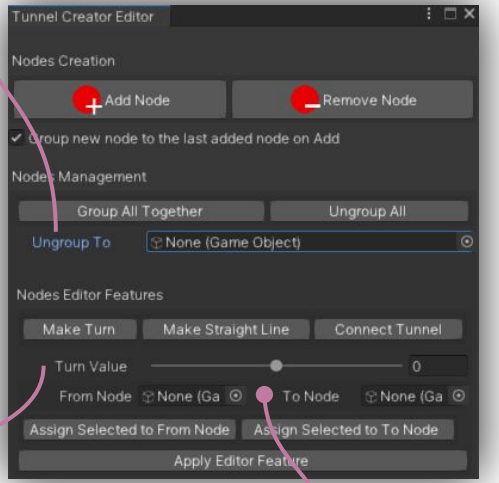
API - *Naming macro **PUBLICtFunct__***

public void PUBLICtFunct_AddNode(Vector3 toPos, bool GroupOnAdd = true)
[Add node on specific position]
public void PUBLICtFunct_RemoveNode()
[Remove last node]
public void PUBLICtFunct_ClearAll()
[Clear all nodes]
public void PUBLICtFunct_ApplyVertexCount()
[Apply changed vertex count and refresh]
public void PUBLICtFunct_RefreshNodes()
[Refresh current tunnel mesh]
public void PUBLICtFunct_GroupNodesTogether()
[Group all nodes together in hierarchy]
public void PUBLICtFunct_UngroupNodes(Transform Detachto)
[Ungroup all nodes to 'empty' or to 'some object']
public void PUBLICtFunct_UpdateUVs()
[Update UV sets with specific UV mode]
public static void GenerateTunnelObj()
[Create brand new gameObject with tunnel creator]

MD_TunnelCreator

Tunnel Creator detailed description.

Tunnel Editor window



Ungroup all nodes to the desired object

Make turn in degrees (Left/Right)

Process 'Turn' from NODE to NODE including all nodes in between these two nodes. (The more nodes included, the smoother the turn evaluates)

Tunnel Vertex count <
Everytime when vertex count is changed, the Apply button must be proceeded & you will lose all your created nodes.
Overall Tunnel Radius <
Overall Node Size <

If enabled, the tunnel will be split into half <
If enabled, the tunnel's normals will be inverted (facing out/in) <
If enabled, the scale of nodes will partially affect tunnel radius <

Currently created nodes (just for debug purposes) <

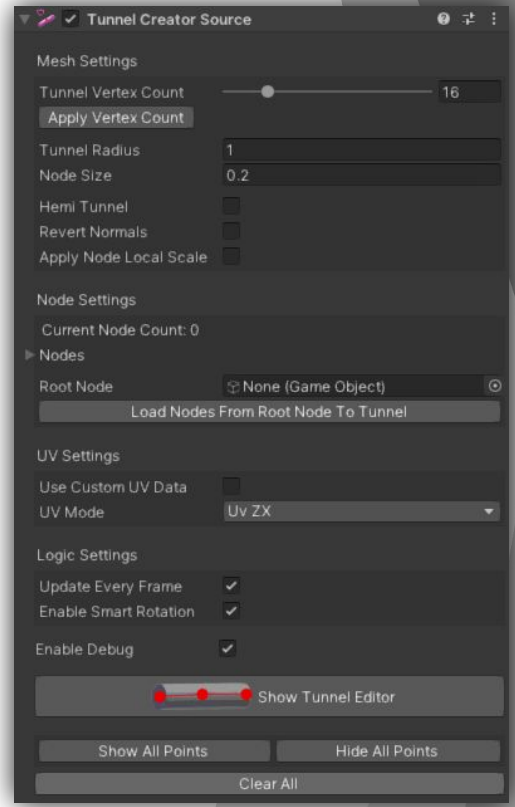
Root node of loaded nodes <
If the button is pressed, the nodes will be automatically loaded into Nodes list. RootNode field must be assigned. This helps you to restore lost tunnels with stored nodes.

Custom UVData option <
If the field is enabled, each tunnel chunk may contain *TunnelNodeUVData* behaviour to customize UV map. Otherwise choose UV_Mode option.

Update Every Frame (takes more performance if bigger tunnels) <
Enable Smart Rotation <
If enabled, the nodes will rotate towards its neighbour. Makes the node editor much easier.

Advanced tunnel editor which helps you to do turns and straight lines a bit easier. <

Inspector view

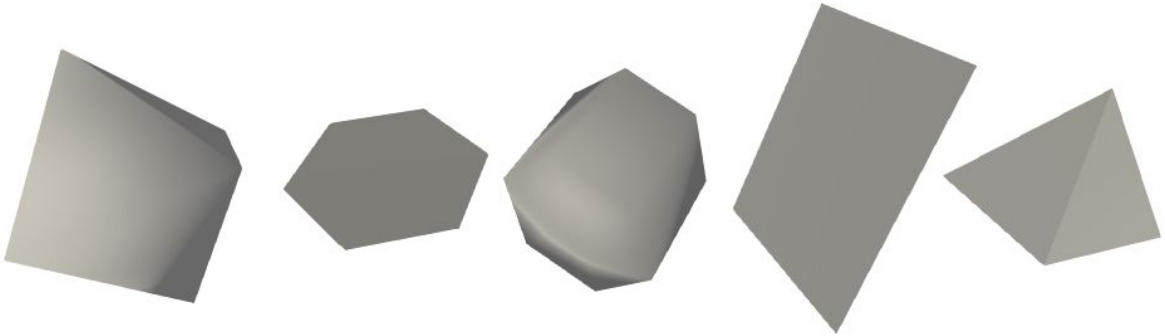


Other Primitives

The Mesh Deformation Package contains other single primitives that can be created in ***Hierarchy/Create 3D/MD Package***.
Create double-sided plane, octahedron, triangle, spatial & planar hexagon.

API

MD_Triangle. Generate()
[Returns generated triangle]
MD_Octahedron. Generate()
[Returns generated octahedron]
MD_TwoSidedPlane. Generate()
[Returns generated two-sided plane]
MD_HexagonGrid. Generate_SinglePlanarHexagon()
[Returns generated planar hexagon]
MD_HexagonGrid. Generate_SingleSpatialHexagon()
[Returns generated spatial hexagon]



Technical Info

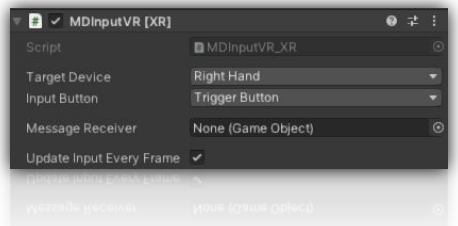
Technical information about the package & other stuff in general

VR Setup

The **MD Package** fully supports all possible VR platforms including Oculus Quest. However, there are some additional steps that you have to make to completely setup the simple VR input for MD Package.

The package contains a folder called **MD_VRPack** which contains all required components for VR setup. There's a folder called *PlatformDependencies* which contains three unity packages. Each package supports specific VR platform that you will have to choose. If you are using SteamVR, import SteamVR package. Same for the other packages.

Once the package is imported, the folder in the root directory will show up with example scenes and **MDInputVR** component. **MDInputVR** is an essential component for simple VR input of the selected platform. It basically tells the VR what button was pressed and sends a message to the desired behaviour. Visit other example scenes for advanced setup of modifiers such as **SculptingPro** or **MeshEditorRuntimeVR**.



If the specified button is pressed, the **MDInputVR** sends a method-message called '*GlobalReceiver_SetControlInput*' to the target object in Message Receiver field. Components that contains this method [MeshEditorRuntimeVR, SculptingPro & MeshPaint] will process the input.

It's a very basic VR input which allows you to record just one button. If you would like to use multiple buttons, you can duplicate components and assign as many buttons as possible (which is not very effective).

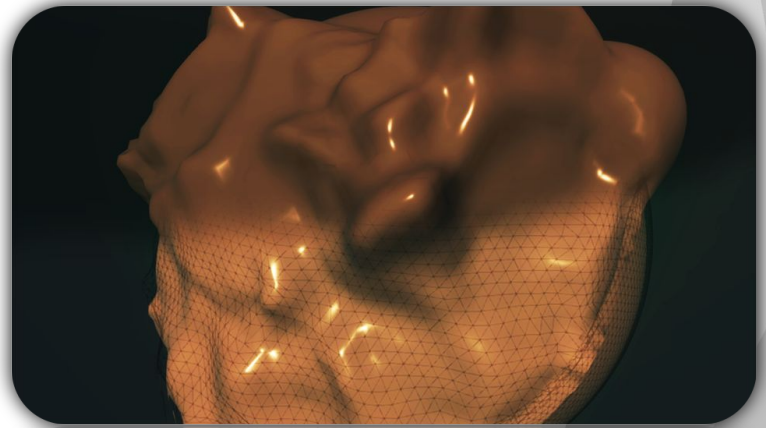
Multithreading

The **MD Package** contains many modifiers that support multithreading. **Multithreading** is an ability to run the specific tasks or calculations on different thread. This allow you to manipulate meshes over 2000 vertex count *(2000 vertex count is a recommended value to prevent performance dropdown).* However, not all modifiers support this feature and you are forced to optimize your mesh or even not to use it for your own safety. It's also recommended not to use many (over 10) modifiers with multithreading feature turned on, this may dropdown your performance. Works well on all devices (PC,VR,Mobile).

Example Video

Car model - 50k vertex count

Planet editor - 16k vertex count



Vertex Tool Window

Vertex Tool Window is an additional feature-tool in MD Package. You can access it in '**Window/MD_Package/VertexTool**' or in the '**Mesh Pro Editor/Vertices Modification**'. It's an expansion for meshes, vertices and elements. Also it contains its own API for internal use called **MD_VertexToolModifier**.

Attach 2 or more meshes [Meshes will be combined and will share the same material]

Clone selected mesh [Parameters below - Count, Position Offset and Rotation Offset]

Weld selected vertices [Vertices will be weld – they will split into one]

Relax selected vertices [Vertices will be normalized and their offset will be multiplied]

Group selected objects into included object below

Group enabled vertices [Additional group function to group enabled vertices in Zone-Generator Mode in MeshProEditor]

Example Video

API for internal use
namespace MD_Plugin

V_TOOL_Element_Attach( )

V_TOOL_Element_Clone()

V_TOOL_Vertex_Weld()

V_TOOL_Vertex_Relax()

The methods above are identical to the window on the left.

Performance & Complexity

MD package has limitless possibilities, but limited performance. That means, it will go harder (not always) with higher-poly meshes with some modifiers & components. Please read the important limitations below that you should know before using the MD Package.

Essential component (MeshProEditor**)**

Mesh Pro Editor allows you to edit mesh vertices directly in the editor as well as at runtime. The component has a condition that if the mesh has more than **N-count vertices** *(specified in preferences)*, you will have to use *Zone-Generator* or you will be forced to optimize your mesh to the lower vertex count. This is not an issue nor feature, as the **MD Package is not focused on advanced Mesh Editor** or vertex editor. Mesh Pro Editor allows you to manipulate with meshes and you can generate its mesh vertices as objects **on a basic level**. If the mesh is beyond the 10 000 vertex count, you won't be able to generate vertices and edit them. This is the major and the only limitation in MeshProEditor. If you would like to edit such meshes (beyond 10 000 vertex count), use multithreaded modifiers such as FFD, MeshEffector, SculptingPro etc.

Modifiers

As mentioned earlier, the package contains multithreaded modifiers that can be used to edit extremely hi-poly meshes. However, this still has a bottleneck. As the Multithreading slide says: having even more multithreaded components may cause performance dropdown, so it's recommended to organize & optimize your project well and use as less as possible multithreaded modifiers.

FAQ

• **Is MD Package available for Mac or Linux?**
Yes, it's available for all operating systems & devices.

• **Can I use MD Package in my mobile game/application?**
Yes, you can, but the mobile performance is very important (depends on goals).

• **Can I edit very complex meshes at runtime/ in editor?**
Yes you can, but it depends which modifier will you choose to work with. Please read Performance & Complexity slide for more info.

• **Is it possible to export my mesh to OBJ format?**
No, it is not possible with MD Package. Use a different external plugin. But you can save your mesh to Assets and create a prefab in the Unity editor.

• **Do I need any programming skills to use MD Package?**
No, you don't need any programming skills. But if you would like to make complex operations, you can use the internal API.

• **Am I able to edit a Skinned Mesh object?**
Yes you can, but not in the way you think. Skinned Mesh Renderers control your mesh by bones. In MD Package the Skinned Mesh Renderer objects have to be converted into Mesh Filters. That means, you will have 2 copies: Original skinned mesh object and editable mesh filter. Original Skinned mesh object will have the mesh source from the editable mesh filter, that means every change made on mesh filter will be applied into the original skinned mesh object.

• **Do I need latest Unity version to use the MD Package?**
No, you don't. But it's much safer and recommended to use the latest Unity Version. If you are going to use the older Unity version, the code conversion and compilation will be required and you can get some warning messages.

• **Is Mesh Tracker included in the Mesh Deformation Full Package?**
No, it's not included in the package. But the package contains similar modifier called Surface Tracking.

• **Is MD Package available for WebGL?**
Yes, but the multithreading method is not allowed so you have to save the performance on your own. [That means, you might experience some performance issues while sculpting high-poly meshes in built game in WebGL]

• **Does MD Package work with Unity HDRP or URP?**
*Yes, **it does**, but **shaders won't work** as they use **Unity standard pipeline**. All the other components and modifiers will work as they are logical part of the package.*

• **Do you plan to make other modifiers *(such as Mesh Damage or FFD)* multithreaded?**
Yes, I do plan making them multithreaded in the future.

FAQ

- **Does MD Package support editing of Unity terrains at runtime?** **...**

No, MD package doesn't work with Unity terrains at all.

Commercial Products

There are some commercial projects that used the MD Package tools and all its features. Explore these projects below. [Click the image]

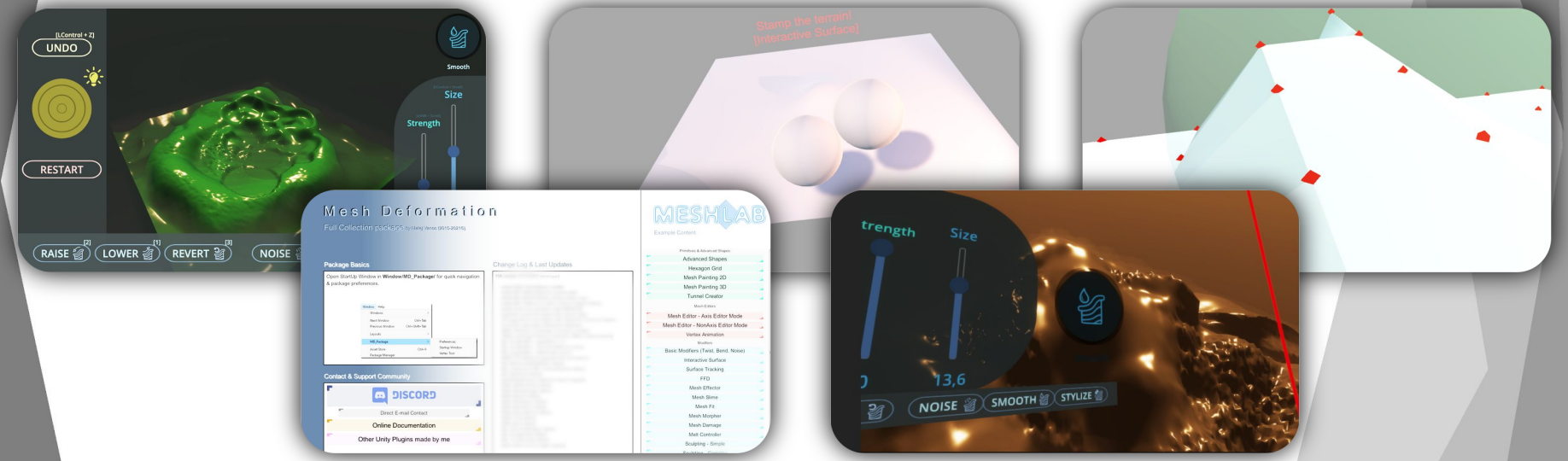
Downloadable Content

You are very free to download official example content of the MD Package.
Available for Windows OS only. [Click the image]

Terrain Sculpting PC [Win]

Modifiers in VR [Win, Unity-XR]

Mesh Editor in VR [Win, Unity-XR]



MD Package official examples [Win]

Terrain Sculpting in VR [Win, Unity-XR]

Extras

The MD Package contains some additional short demo-games that were built from the package modifiers and shaders. Explore these games for free below! Just click the gif image.

Thank you!

Thank you for your attention. I hope you have become more experienced. If you have any questions, suggestions or issue reports, do not hesitate and join to my official discord channel for **quick** & **realtime** support.

If you don’t like Discord, you can still contact me **here**.