Canvas Unity3D hoạt động như thế nào?

UI, Canvas, Layout, Image,… chắc mọi người không còn xa lạ gì, mình cũng sẽ không nói lại cách sử dụng các thứ nữa. Mặc định là các bạn đã biết hết rồi nhé :>

Mục đích của bài chỉ để trả lời 2 câu hỏi mình vẫn hay thắc mắc khi sử dụng UI, Graphics system vẽ UI như thế nào? Có khác vẽ GameObject (không phải UI) khác không? Nếu hiểu rồi thì có optimized được chỗ nào không?
Thực ra là 3 câu hỏi.

1. Canvas và ông họa sỹ
Canvas là nơi đặt các component UI lên, là khung hỗ trợ vẽ UI, là vùng chứa UI, là …, vân vân và mây mây. Vậy thực chất “nó” là gì?

Đầu tiên, Canvas sẽ tìm các child có CanvasRenderer (component hỗ trợ vẽ Image và Text)

Canvas kết hợp những khối hình giống nhau về setting như material, sprite,… của các đối tượng con vào thành những khối gọi là batches, sau đó tạo các lệnh “Render Commands” gửi cho Graphics System của Unity, bắt Graphics vẽ theo setting của các batches.

Ngoài ra những thứ này đều được xử lý bằng C++ nên rất tối ưu.

Batching Technique

“Tại sao lại có batch/ batches gì ở đây?”

Khi sử dụng kỹ thuật batching hay chia sẻ settings giữa nhiều objects, Graphics sẽ vẽ các objects cùng batch đó đúng trong một lần.

Ví dụ một lão họa sỹ đang lên màu cho một bức tranh, trong tay ổng là chiếc bút màu đỏ, để hoàn thành nhanh chóng bức tranh bằng kỹ thuật batching, ổng sẽ tô hết tất cả các vật thể bằng màu đỏ trước, chứ ổng không tô lần lượt: đỏ (vật A) -> xanh (vật B) -> vàng (vật C) -> đỏ (vật D) -> tím ->… et cetera.
Ở đây batches là “setting màu đỏ”, các vật thể (CanvasRenderer) nằm trên trang giấy (canvas), còn ông họa sỹ đóng vai trò như Graphics

“Vậy những khối hình mà Canvas dùng để kết hợp lấy ở đâu?”

Khi chúng ta thêm một component như Image, Text,… sẽ đi kèm với một component khác, đó là CanvasRenderer như mình đã đề cập ở trên.

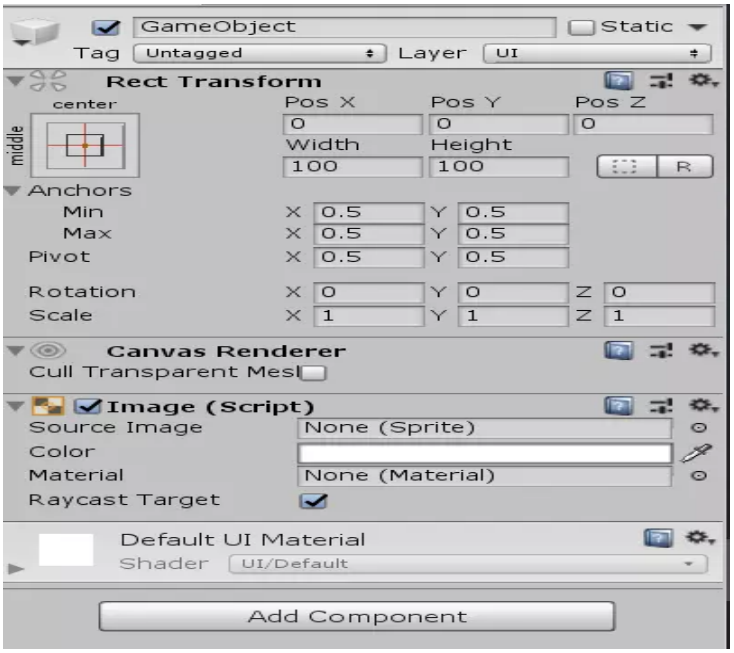


Image component requires Canvas Renderer
“Vậy khi nào thì Canvas sẽ batching lại (rebatch hay batch build)?”

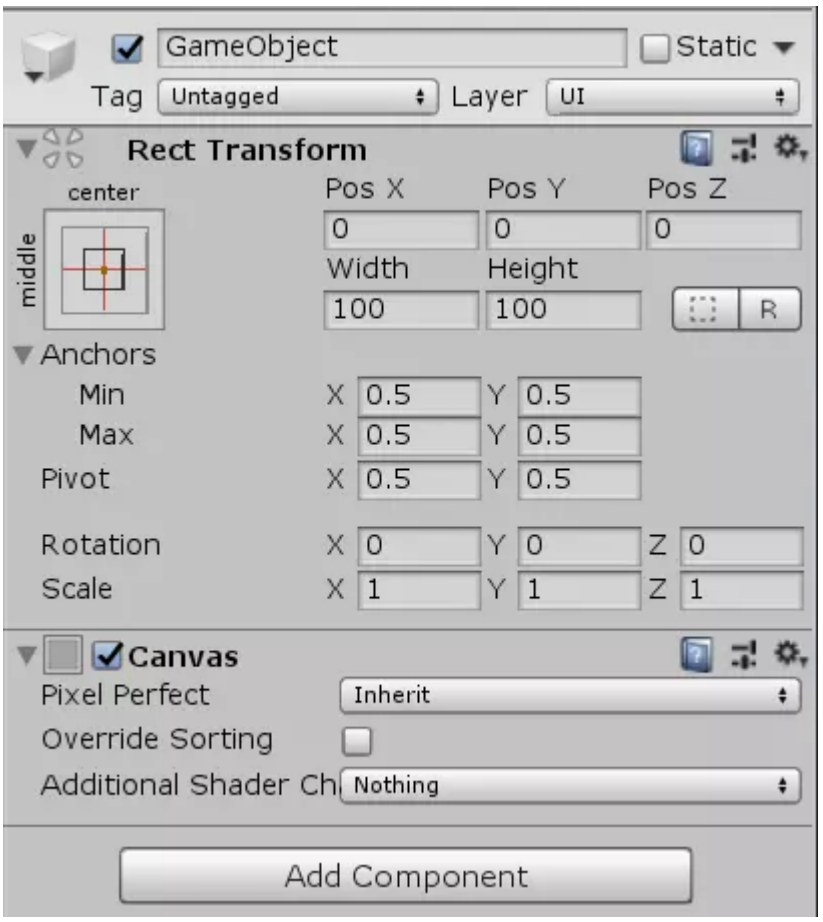
Khi một trong những khối hình cung cấp bởi cácCanvasRenderer (child component) có sự thay đổi, có nghĩa là, chỉ cần một sự thay đổi nhỏ ở các đối tượng con của nó đồng nghĩa với việc Canvas sẽ bị đánh dấu là DIRTY (dirty flag) và phải rebatch lại toàn bộ

Để tối ưu (optimize) vấn đề này, người ta thường sử dụng một thứ gọi là Sub-canvas hay nested-canvas, vậy Sub-canvas là cái quái gì?

Nếu các bạn hứng thú cách mà CPU bảo Graphics vẽ như nào thì đây: Cách GPU vẽ lên màn hình

2. Sub-canvas hay Nested-canvas
NestedCanvas đơn giản là một Canvas nằm trong một Canvas khác

subCanvas sẽ tách các child-objects của nó khỏi parentCanvas, sự thay đổi của child objects trong subCanvas sẽ đánh dấu subCanvas này là DIRTY, không ảnh hưởng tới Canvas cha và ngược lại.



Sub-canvas hay nested canvas
Tuy nhiên có một số ngoại lệ, nếu việc thay đổi ở Canvas cha ảnh hưởng tới SubCanvas thì cả 2 sẽ đều bị đánh dấu DIRTY, ví dụ thay đổi size của Canvas cha làm canvas con phải scale theo.

Mặc dù optimized hơn nhưng vẫn có một sự đánh đổi (trade-off) nhẹ ở đây, khi sử dụng Sub-canvas, mặc dù số lần rebatch giảm nhưng số lượng batches sẽ tăng lên, vì vậy không phải trường hợp nào cũng sử dụng cách này.

3. Graphic
Class Graphic này cung cấp cho Image và Text,… khả năng “được vẽ”.
Lưu ý là từ khóa Graphic khác với Graphics.

public class Image : MaskableGraphic, ISerializationCallbackReceiver, ILayoutElement, ICanvasRaycastFilter

public class Text : MaskableGraphic, ILayoutElement

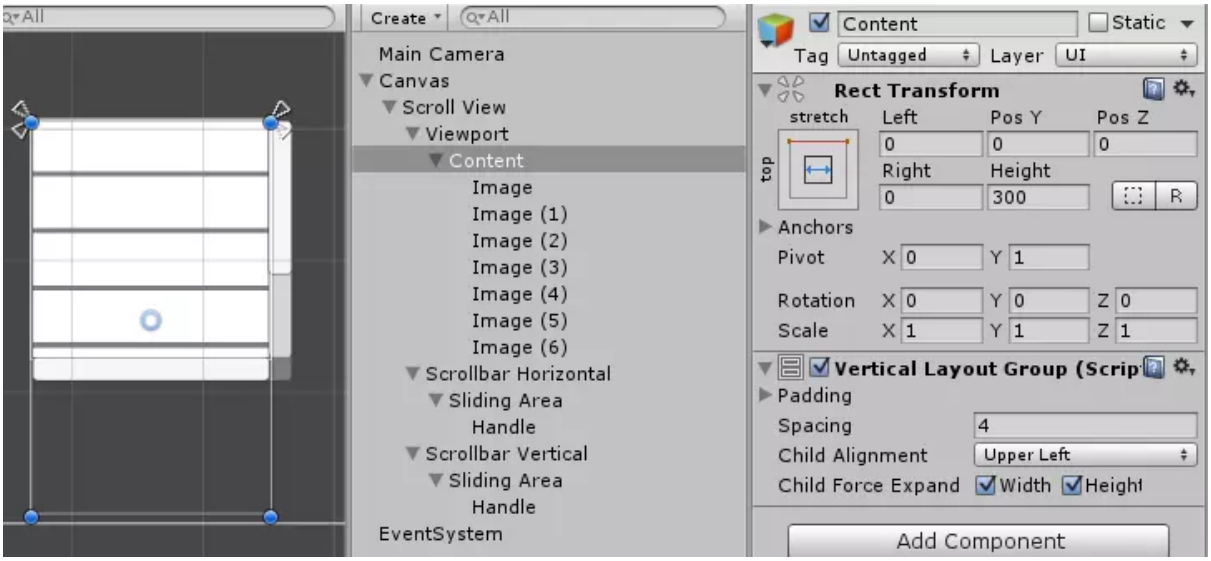
public abstract class MaskableGraphic : Graphic, IClippable, IMaskable, IMaterialModifier

Ở đây, class Image, Text kế thừa MaskableGraphic, MaskableGraphic lại kế thừa Graphic

Đa phần các UI Graphics trong Unity được implemented từ MaskableGraphic, nhiều bạn sử dụng RectMask2D hay Mask thì có thể biết chức năng mask rồi nên mình không nói thêm nhé.

4. Layout
Vertical/ Horizontal/ Grid Layout Group… là các thành phần UI không hỗ trợ vẽ, không kế thừa MaskableGraphic, Graphic như Image hay Text.

Nhưng chúng cũng sở hữu sức mạnh quan trọng trong thiết kế UI đó là khả năng điều khiển kích thước (size), vị trí (position) của child-RectTransform theo một trật tự, logic mong muốn như sắp xếp theo cột, theo bảng, theo hàng…



Ảnh minh họa Layout
5. CanvasUpdateRegistry
Cả 2 lớp Graphic và Layout đều phụ thuộc vào lớp CanvasUpdateRegistry, trong Unity Editor sẽ không thấy lớp này.

CanvasUpdateRegistry sẽ theo dõi các Layout components và Graphic components nào cần phải được cập nhật hay tính toán lại. Sau đó thực hiện việc cập nhật khi Canvas quản lý các components này thực thi sự kiện (event) willRenderCanvases để vẽ lại, event này sẽ được gọi vào mỗi frame.

Đồng thời, quá trình cập nhật hay rebuild lại của component Graphic và Layout cũng khác nhau, các giai đoạn rebuild cũng khác nhau, các bạn có thể xem thêm về class CanvasUpdateRegistry ở phần Quy trình rebuild ở dưới.

6. Quy trình Batch (Canvases)
Batching Process là quá trình Canvas lấy các meshes được cung cấp bởi CanvasRendener con (không tính subCanvas), kết hợp các meshes có cấu hình giống nhau thành một batch, sau đó tạo các Rendering Commands (lệnh render) gửi tới bộ phận Graphics.

Kết quả của quá trình này trong 1 frame sẽ được lưu trữ (cached) và sử dụng cho các frames tiếp theo cho tới khi Canvas bị đánh cờ DIRTY (dirty flag), nó bị đánh cờ DIRTY khi một trong các meshes được cung cấp bởi CanvasRenderer con có sự thay đổi, về kích thước, sprite, vị trí, scale, rotation,…

Batching Process thường thực thi trên các threads khác (multithreading), vì vậy performance sẽ có sự khác biệt tùy theo số lượng core hoặc processor của các máy multicore processor, multi-processors.

7. Quy trình Rebuild (Graphics)
Rebuild Process là quá trình tính toán lại layout và meshes của các thành phần Graphic (Text, Image,…), Layout (Grid, HorizontalLayoutGroup, VerticalLayoutGroup…). Việc tính toán này được thực hiện trong class singleton CanvasUpdateRegistry mà mình đã đề cập ở phần trên. Thông qua hàm CanvasUpdateRegistry.PerformUpdate() được gọi từ event Canvas.willRenderCanvases (event này được gọi vào mỗi frame)

“Class Singleton là gì?”

Các bạn có thể tham khảo ở Có nên sử dụng Singleton Pattern?

```
public class CanvasUpdateRegistry
{
	protected CanvasUpdateRegistry()
	{
	   Canvas.willRenderCanvases += PerformUpdate;
	}
	private void PerformUpdate()
	{
	}
}
```

“Làm thế nào CanvasUpdateRegistry biết các thành phần Graphic hay Layout nào cần rebuild?”

CanvasUpdateRegistry có 2 queues bao gồm: queue “các layouts cần rebuild” và queue “các graphics cần rebuild”, các dirty-layout và dirty-graphic sẽ đăng ký vào 2 queues này thông qua ICanvasElement.Rebuild

```
public class CanvasUpdateRegistry
{
   private readonly IndexedSet<ICanvasElement> m_LayoutRebuildQueue = new
IndexedSet<ICanvasElement>();

   private readonly IndexedSet<ICanvasElement> m_GraphicRebuildQueue = new
IndexedSet<ICanvasElement>();

   private bool InternalRegisterCanvasElementForLayoutRebuild(ICanvasElement element)
   {
```

```
        if (m_LayoutRebuildQueue.Contains(element))
            return false;
        return m_LayoutRebuildQueue.AddUnique(element);
    }

    private bool InternalRegisterCanvasElementForGraphicRebuild(ICanvasElement element)
    {
        if (m_PerformingGraphicUpdate)
            return false;
        return m_GraphicRebuildQueue.AddUnique(element);
    }
}
```