Flutter

Union-typed transform stack in hit test

SUMMARY

Push various types besides matrices (such as offsets) onto the transform stack during hit testing to reduce overhead.

Author: Tong Mu (dkwingsmt)
Go Link: flutter.dev/go/union-typed-transform-stack-during-hit-test
Created: 6/2020 / **Last updated:** 6/2020

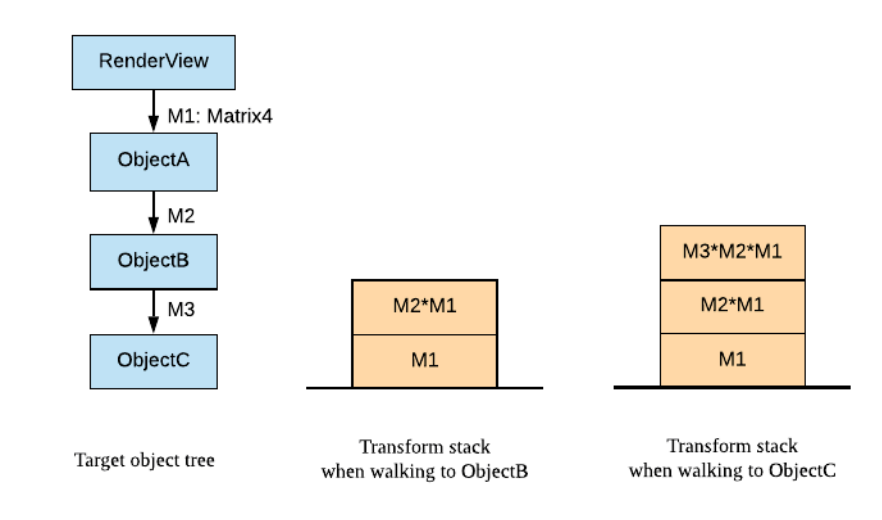
OBJECTIVE

Reduce the overhead during hit testing, allowing mouse hit testing to use the regular hit testing system without performance regression.

BACKGROUND

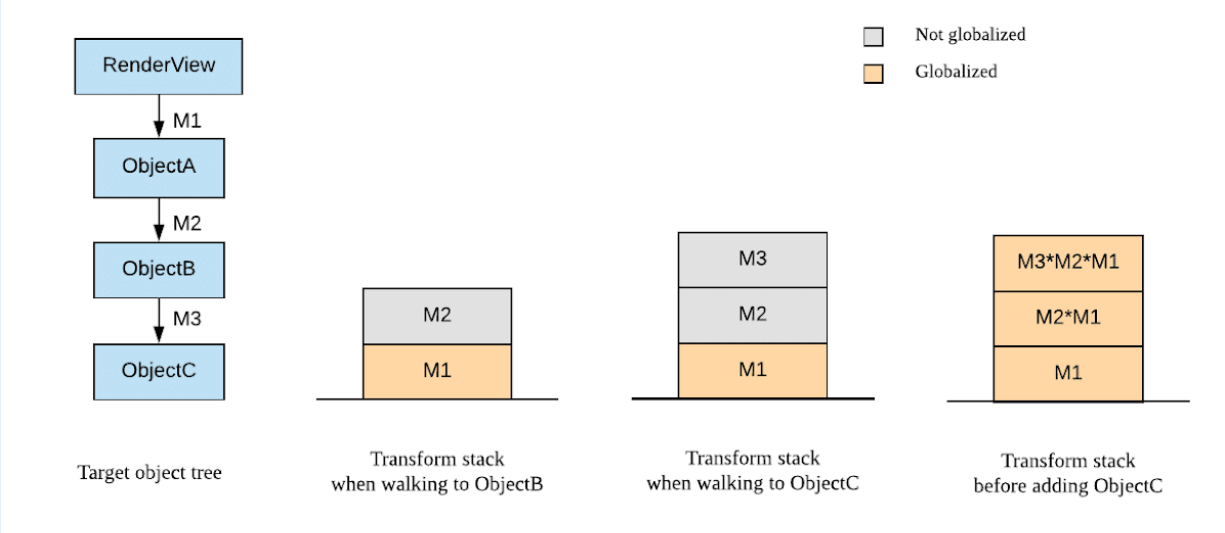
An on-going project **Direct Mouse Hit Testing** aims to change **mouse hit testing** from the current layer-based approach to the **regular hit testing system**. This will make the system much simpler, but requires further optimization of the hit testing system.

Profiling revealed that a bottleneck during the hit test is calculating the transform matrices to be pushed onto the transform stack, which are used for the render objects to convert global locations to their local coordinate spaces.



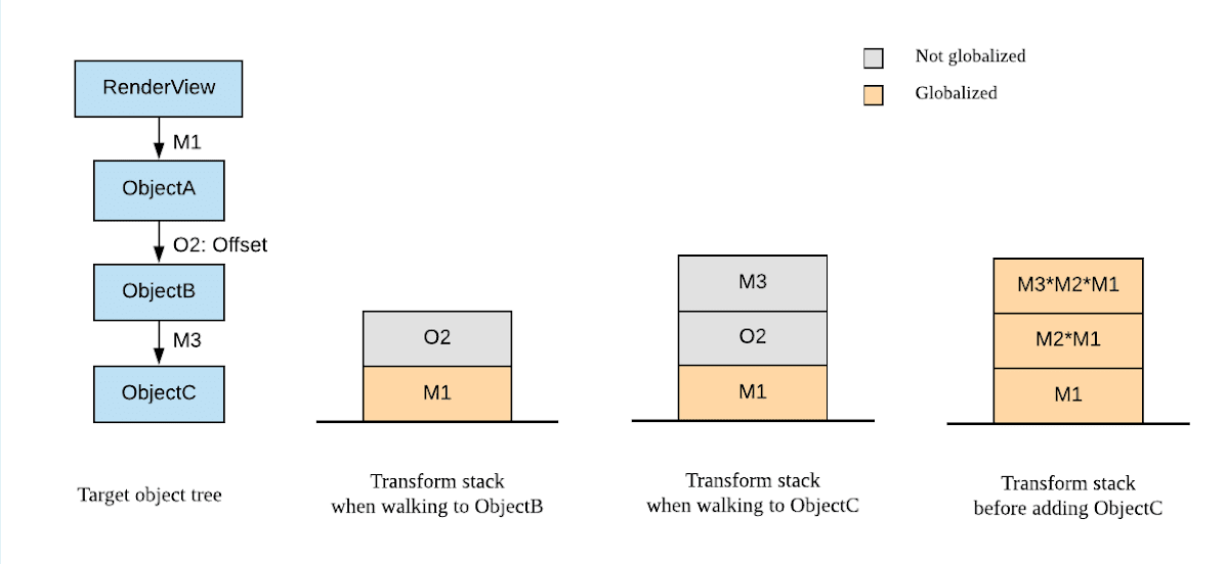
Transform stacks as in the current hit test system

The first change is **lazy transform multiplication**: to push the **transform parts** (i.e. local transform matrices) onto the stack, and only multiply them when necessary (right before the object is added). This has shown a ~30% frame time difference. This change is implementation-only and requires no API change.



Transform stacks with lazy transform multiplication

The second change is **union-typed transform stack**: to push non-matrices onto the stack. Translations (offsets) constitute most of the transforms pushed, and while they can be applied to matrices in an easier way, they're still converted to general matrices and applied by multiplication. If offsets can be pushed onto the same stack along with matrices, they can be applied using the much cheaper `Matrix4.leftTranslate`. This requires that the transform stack takes a union type of `Matrix4` and `Offset`, hence the name.



Transform stacks with union-type transform parts

Benchmarks have shown that this change can allow direct mouse hit testing to catch up with, or even outperform the layer-based one.

Average frame time of `bench_mouse_region_grid_hover` (Web)	
(a) Layer-based	2.8ms
(b) Direct, with some optimization	7.7ms
(c) Direct, with (b) as well as offset pushing	1.7ms

However, this change requires API change, because `HitTestResult` only allows pushing transform *matrices*.

Glossary

- **(Regular) Hit testing** - A depth-first walk over the render object tree to find a list of objects that contains the target position. Used to dispatch pointer events (such as tapping).
- **Mouse hit testing, layer based** - A hit testing used to dispatch mouse events (such as mouse enter and exit). Currently it has been running on the layer tree due to higher performance requirements, hence being *layer-based*.
- **Direct mouse hit testing** - Use the regular hit testing system to do mouse hit testing. Also refers to the project to investigate optimization to bring the performance on par.
- **Transform part** - The local transform matrix of a render object from its parent, as opposed to the global transform matrix from the root.

OVERVIEW

- `HitTestResult` has new methods `pushTransformPart` and

`popTransformPart`, replacing the current `pushTransform` and `popTransform`.

- `TransformPart` is a new class that stands for any data that can be left-multiplied to a matrix.

Non-goals

- Implementing lazy transform multiplication. This change should be considered done in the context of this design doc, although not landed yet.

DETAILED DESIGN/DISCUSSION

Currently pushing an offset to `BoxHitTestResult` takes `addWithPaintOffset`, which calls `addWithRawTransform`, which calls `pushTransform`.

```
// CURRENT VERSION WITH SLIGHT MODIFICATION FOR CLARITY

class BoxHitTestResult extends HitTestResult {
  /// Convenience method for hit testing children, that are translated by
  /// an [Offset].
  bool addWithPaintOffset({
    @required Offset offset,
    @required Offset position,
    @required BoxHitTest hitTest,
  }) {
    return addWithRawTransform(
      transform: offset != null ? Matrix4.translationValues(-offset.dx, -offset.dy,
0.0) : null,
      position: position,
      hitTest: hitTest,
    );
  }

  /// Transforms `position` to the local coordinate system of a child for
  /// hit-testing the child.
  bool addWithRawTransform({
    @required Matrix4 transform,
    @required Offset position,
    @required BoxHitTest hitTest,
  }) {
    final Offset transformedPosition = ...;
    if (transform != null) {
      pushTransform(transform);
    }
    final bool isHit = hitTest(this, transformedPosition);
    if (transform != null) {
      popTransform();
    }
    return isHit;
  }

  // INHERITED FROM [HitTestResult] //
```

```
  // Changes compared to master: `_transforms` has been changed from
  // Queue to List.
  final List<Matrix4> _transforms;

  /// Pushes a new transform matrix that is to be applied to all future
  /// [HitTestEntry]s added via [add] until it is removed via [popTransform].
  @protected
  void pushTransform(Matrix4 transform) {
    // Changes compared to master: the pushed transform has been changed to
    // perform lazy multiplication .
    _transforms.add(transform);
  }
}
```

We want `addWithPaintOffset` to push an offset without converting it to a matrix first, but it can't be done with only `pushTransform`.

Two new methods will be added, which push and pop a new class `TransformPart`, deprecating the current two that push and pop `Matrix4`.

```
class BoxHitTestResult extends HitTestResult
{
  bool addWithPaintOffset({
    @required Offset offset,
    @required Offset position,
    @required BoxHitTest hitTest,
  }) {
    // Changed to directly pushing an offset onto the transform stack
    final Offset transformedPosition = ...;
    if (offset != null) {
      pushTransformPart(OffsetTransformPart(offset));
    }
    final bool isHit = hitTest(this, transformedPosition);
    if (offset != null)
    {
      popTransformPart();
    }
    return isHit;
  }

  bool addWithRawTransform({
    @required Matrix4 transform,
    @required Offset position,
    @required BoxHitTest hitTest,
  }) {
    final Offset transformedPosition = ...;
    if (transform != null)
    {
      // Changed to using the new methods
      pushTransformPart(TransformPart.matrix(transform));
    }
```

```
    final bool isHit = hitTest(this, transformedPosition);
    if (transform != null)
    {
      // Changed to using the new methods
      popTransform();
    }
    return isHit;
  }

  // INHERITED FROM [HitTestResult] //

  // Stack elements changed from Matrix4 to TransformPart
  final List<TransformPart> _transforms;

  // New methods
  @protected
  void pushTransformPart(TransformPart transform)
  {
    _transforms.add(transform);
  }
  @protected
  void popTransformPart()
  {
    _transforms.removeLast();
  }

  // Deprecate methods
  @deprecated
  @protected
  void pushTransform(Matrix4 matrix);
  @deprecated
  @protected
  void popTransform();
}
```

`TransformPart` is defined as any data that can be left-multiplied to a matrix, producing a new matrix. Apparently a matrix is a `TransformPart`, although the class is hidden to ~~reduce~~ a public class, and because matrix is the core of this class.
¶

```
@immutable¶
abstract class TransformPart ¶
{¶
  const TransformPart();¶
¶
  factory TransformPart.matrix(Matrix4 matrix) => _MatrixTransformPart(matrix);¶
¶
  Matrix4 _assertMatrix() {¶
    assert(false, '$this is not a Matrix transform part.');¶
    throw UnimplementedError('$this is not a Matrix transform part.');¶
```

```
  }¶
¶
  TransformPart multiply(Matrix4 rhs);¶
}¶
¶
class _MatrixTransformPart extends TransformPart ¶
{¶
  const _MatrixTransformPart(this.matrix);¶
¶
  final Matrix4 matrix;¶
¶
  @override¶
  Matrix4 _assertMatrix() => matrix;¶
¶
  @override¶
  TransformPart multiply(Matrix4 rhs) ¶
  {¶
    return TransformPart.matrix(matrix * rhs as Matrix4);¶
  }¶
}¶
```

An offset is also a `TransformPart`.

```
class OffsetTransformPart extends TransformPart {
  const OffsetTransformPart(this.offset);

  final Offset offset;

  @override
  TransformPart multiply(Matrix4 rhs) {
    return TransformPart.matrix(rhs.clone()..leftTranslate(offset.dx, offset.dy));
  }
}
```

ALTERNATIVE SOLUTIONS

1. **Change pushTransform to take `Object` as argument** with runtime type check
2. **A new method `HitTestResult.pushOffset`** (possibly with `.popOffset`)
3. **Implement `Matrix4` that wraps a translation offset into a matrix** (all 90 methods!)

TESTING PLAN

- All current unit tests should pass
- Some new unit tests to verify the extended flexibility

MIGRATION PLAN

The legacy methods will be marked deprecated now, and removed in the future.