Surf

surf_tech

Кастомим пейнтер через Custom Painter

Илясов Айнур
TeamLead, Flutter Dev

Я

surf_tech

- Айнур Ильясов
- Senior Flutter Developer
- TeamLead
- DevRel Ambassador

в SURF

О чем будем говорить?

surf_tech

1

База

2

Canvas

3

Анимация

4

Интерактивность

5

Оптимизация

6

Лееринг

Часть 1

Это база

База

surf_tech

Custom Painter - это то, с помощью чего можно взаимодействовать с напрямую с **Canvas**

Для чего нужен

1. Нехватает инструментов в Flutter из коробки;
2. Оптимизация сложной отрисовки;

Для чего нужен

surf_tech

- Custom Paint
- Render Object

Custom Paint

Это widget для отрисовки **CustomPainter**.

Параметры:

- ***painter*** - отрисовывается до ребенка
- ***foregroundPainter*** - отрисовывается после ребенка
- ***size*** - размеры, если не передали ребенка
- ***isComplex*** - включение кеширования слоев для тяжеловесной графики
- ***willChange*** - отключение кеширования для растрового слоя этого виджета

Custom Paint

surf_tech

```
class CustomPaint extends SingleChildRenderObjectWidget
```

Custom Paint

surf_tech

```dart
@override
void paint(PaintingContext context, Offset offset) {
  if (_painter != null) {
    _paintWithPainter(context.canvas, offset, _painter!);
    _setRasterCacheHints(context);
  }
  super.paint(context, offset);
  if (_foregroundPainter != null) {
    _paintWithPainter(context.canvas, offset, _foregroundPainter!);
    _setRasterCacheHints(context);
  }
}
```

CustomPainter

surf_tech

```
class MyCustomPainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {}

  @override
  bool shouldRepaint(MyCustomPainter oldDelegate) ⇒ false;

  @override
  bool shouldRebuildSemantics(MyCustomPainter oldDelegate) ⇒ false;
}
```

CustomPainter

- ***paint*** - место для отрисовки. Принимает **Canvas** и **Size**
- ***hitTest*** - отслеживаем жесты
- ***shouldRepaint*** - в какой момент вызвать paint еще раз
- ***shouldRebuildSemantics*** - обновление дерева семантики

Paint


```dart
@override
void paint(Canvas canvas, Size size) {
  final paint = Paint();
}
```

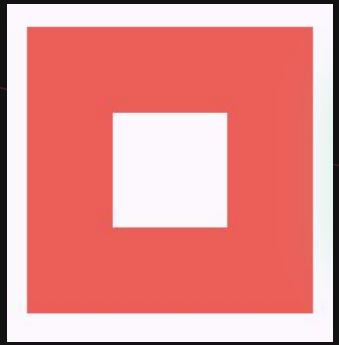
Paint

Параметры:

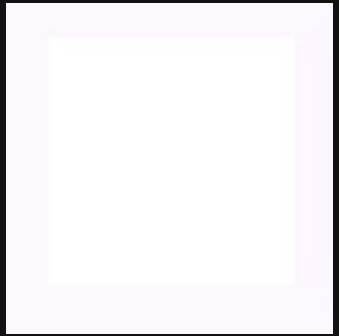
- ***color*** - цвет
- ***strokeWidth*** - ширина кисти
- ***style*** - fill/stroke

Paint


```dart
final paintStroke = Paint()
  ..color = Colors.redAccent
  ..strokeWidth = 30
  ..style = PaintingStyle.stroke;
```



```dart
final paint = Paint()
  ..color = Colors.white
  ..strokeWidth = 10
  ..style = PaintingStyle.fill;
```



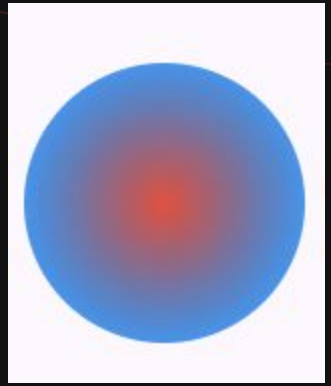
Paint

Параметры:

- *shader*

Paint


```dart
final paintFill = Paint()
  ..color = Colors.blueAccent
  ..shader = const RadialGradient(
    colors: [Colors.red, Colors.blue],
  ).createShader(Rect.fromCircle(
    center: Offset(50, size.height / 2),
    radius: 35,
  ))
  ..style = PaintingStyle.fill;
```



Paint


```
final paint = Paint()
  ..shader = ImageShader(
    image,
    TileMode.repeated,
    TileMode.repeated,
    Matrix4.identity().storage,
  );
```

Paint


```
final paint = Paint()
  ..shader = ui.FragmentShader();
```

Canvas

Это “холст”, предоставляющий набор **инструкций** отрисовки.

Инструкции:

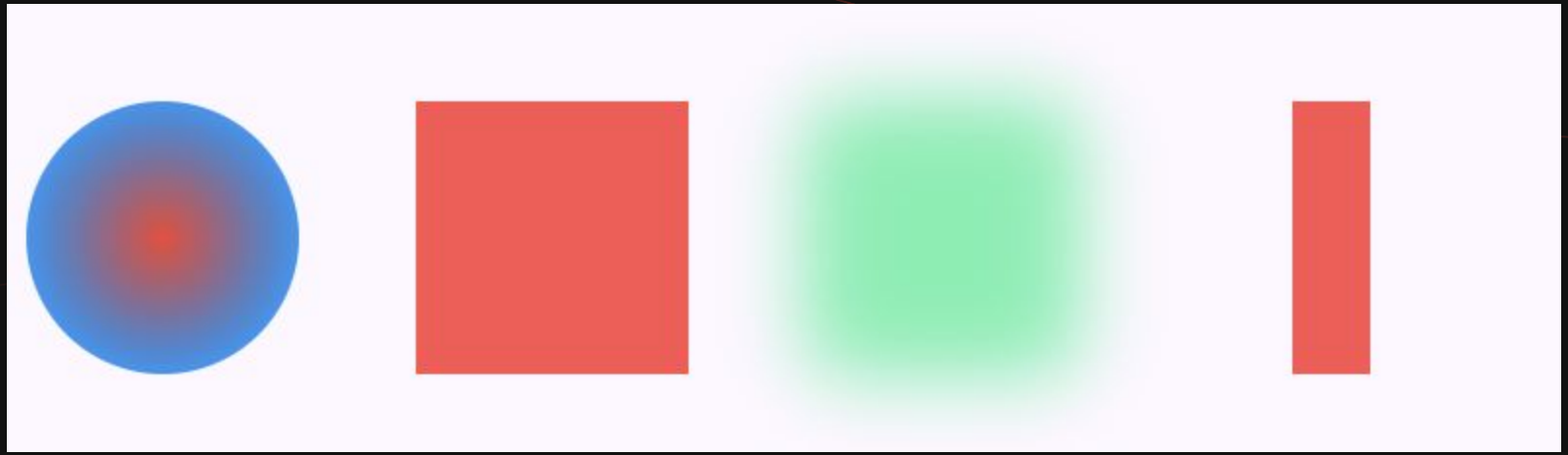
- Рисование примитивов;
  - drawCircle
  - drawRect
  - drawLine
- Рисование путей
- Рисование групп объектов
- Методы для трансформации

Инструкции Canvas

surf_tech

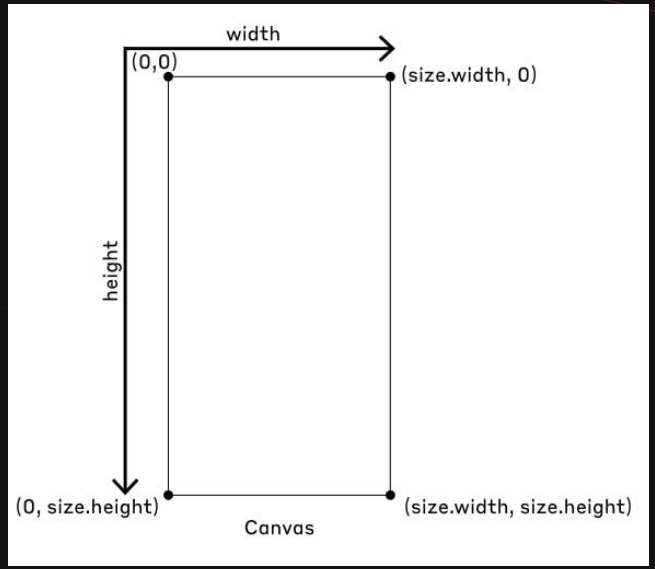
```
canvas.drawCircle(
  Offset(50, size.height / 2),
  figureSize / 2,
  paintFill,
);

canvas.drawRect(
    Rect.fromCenter(
      center: Offset(150, size.height / 2),
      width: figureSize,
      height: figureSize,
    ),
    paintStroke);

canvas.drawRRect(
  RRect.fromRectAndRadius(
      Rect.fromCenter(
        center: Offset(250, size.height / 2),
        width: figureSize,
        height: figureSize,
      ),
      const Radius.circular(10)),
  paintBlur,
);

canvas.drawLine(
  Offset(350, size.height / 2 - figureSize / 2),
  Offset(350, size.height / 2 + figureSize / 2),
  paintLine
);
```

Инструкции Canvas



Offset

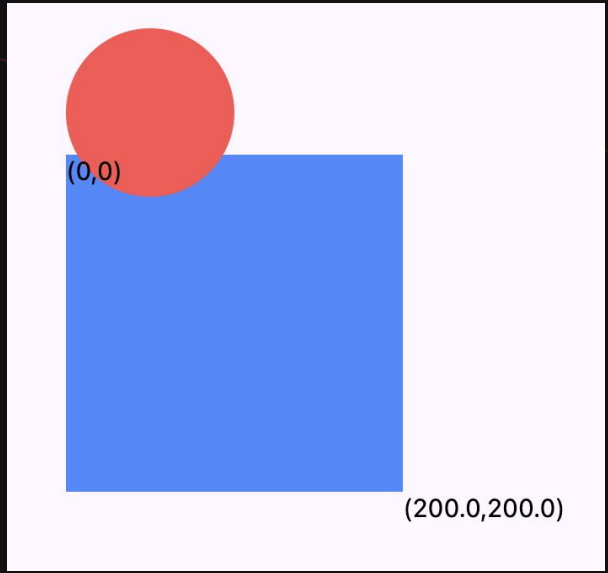
Смещение от точки (0,0)



Offset

surf_tech

```
canvas.drawRect(
  Rect.fromCenter(
    center: Offset(size.width / 2, size.height / 2),
    width: size.width,
    height: size.height,
  ),
  paintFill,
);
canvas.drawCircle(const Offset(50, -25), 50, paintCircle);
```



Path

Это набор инструкций, который сформирован в одну фигуру.

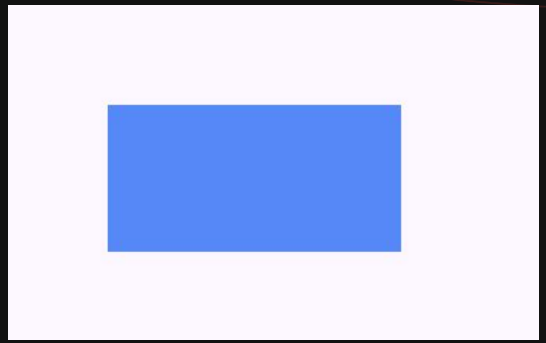
Инструкции:

- Рисование примитивов;
  - addCircle
  - addRect
  - lineTo
- Кривые безье
- Абсолютное/относительное смещение

Path

surf_tech

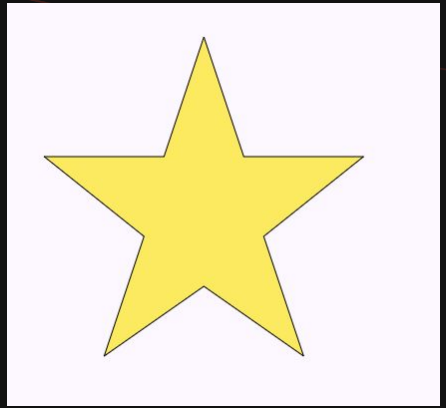
```dart
final path1 = Path()
    ..moveTo(50, size.height / 2)
    ..lineTo(150, size.height / 2)
    ..lineTo(150, size.height / 2 + 50)
    ..lineTo(50, size.height / 2 + 50)
    ..close();
```



Path

surf_tech

```dart
void paint(Canvas canvas, Size size) {
  final paintFill = Paint()
    ..color = Colors.yellow
    ..style = PaintingStyle.fill;

  final paintStroke = Paint()
    ..color = Colors.black
    ..style = PaintingStyle.stroke;

  final path = Path()
    ..moveTo(100, 10)
    ..relativeLineTo(20, 60) // → (120, 70)
    ..relativeLineTo(60, 0) // → (180, 70)
    ..relativeLineTo(-50, 40) // → (130, 110)
    ..relativeLineTo(20, 60) // → (150, 170)
    ..relativeLineTo(-50, -35) // → (100, 135)
    ..relativeLineTo(-50, 35) // → (50, 170)
    ..relativeLineTo(20, -60) // → (70, 110)
    ..relativeLineTo(-50, -40) // → (20, 70)
    ..relativeLineTo(60, 0) // → (80, 70)
    ..close();

  canvas.drawPath(path, paintFill);
  canvas.drawPath(path, paintStroke);
}
```



Path. Кривые безье

surf_tech

Это кривая, с помощью которой можно плавно соединить точки.
Используется для сглаживаний, скруглений итд.

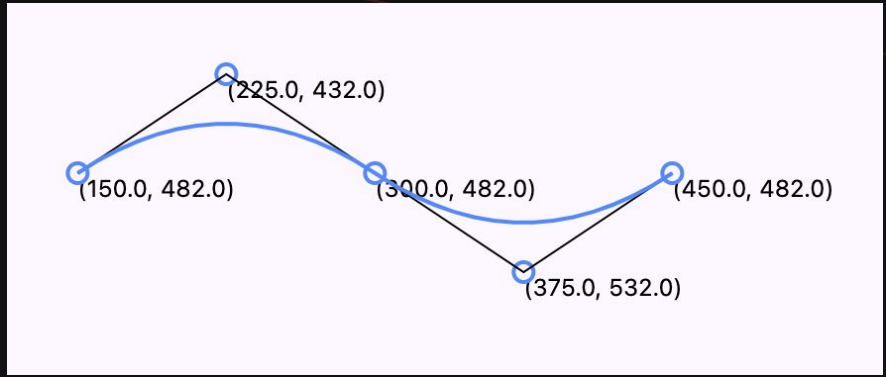
Методы для отрисовки:

- quadraticBezierTo
- cubicTo
- conicTo

Path. Кривые безье

surf_tech

```dart
final path = Path()

  /// опорная точка
  ..moveTo(150, size.height / 2)
  ..quadraticBezierTo(
    /// управляющая точка
    225,
    size.height / 2 - 50,

    /// опорная точка
    300,
    size.height / 2,
  )
  ..quadraticBezierTo(
    /// управляющая точка
    375,
    size.height / 2 + 50,

    /// опорная точка
    450,
    size.height / 2,
  );
```



Canvas. Состояние

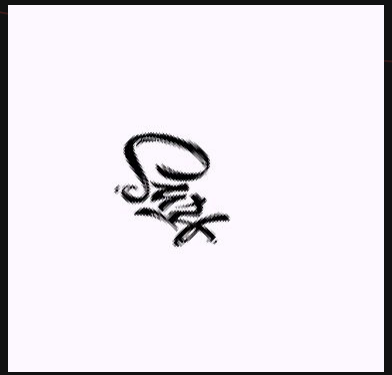
surf_tech

У нашего хоста есть свое состояние:

- трансформации (scale, rotate, translate);
- clip - обрезка;
- Альфа-прозрачность.

Canvas. Состояние

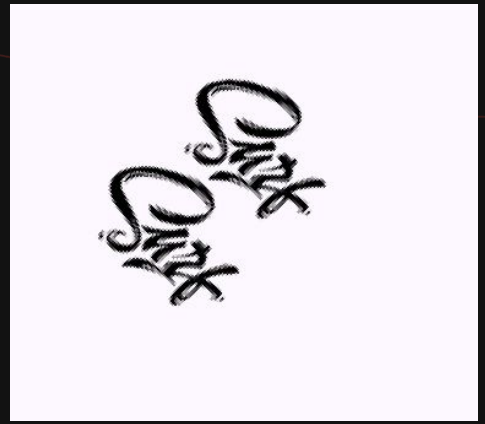

```
canvas.rotate(math.pi * 1 / 4);
canvas.drawImage(image, const Offset(200, 0), Paint());
```



Canvas. Состояние

surf_tech

```
canvas.rotate(math.pi * 1 / 4);
canvas.drawImage(image, const Offset(200, 0), Paint());
canvas.drawImage(image, const Offset(200, 50), Paint());
```



Canvas. Состояние

surf_tech

```
canvas.save();
canvas.rotate(math.pi * 1 / 4);
canvas.drawImage(image, const Offset(200, 0), Paint());
canvas.restore();
canvas.drawImage(image, const Offset(200, 50), Paint());
```

Canvas. Состояние


```
canvas.save();
canvas.rotate(math.pi * 1 / 4);
canvas.drawImage(image, const Offset(200, 0), Paint());

canvas.save();
canvas.scale(2);
canvas.translate(40, 40);
canvas.clipRect(
  Rect.fromCenter(
    center: size.centerOffset,
    width: size.half.width,
    height: size.half.height,
  ),
);
canvas.drawImage(image, const Offset(200, 50), Paint());

canvas.restoreToCount(2);
```

Часть 2

Анимации

Анимации

surf_tech

Что нам надо для анимации?

- AnimationController
- CustomPainter

Анимации


```
return AnimatedBuilder(
  animation: _controller,
  builder: (_, _) {
    return CustomPaint(
      painter: AnimatedPainter(
        value: _animation.value,
      ),
    );
  },
);
```

Анимации


```
const CustomPainter({Listenable? repaint}) : _repaint = repaint;
```

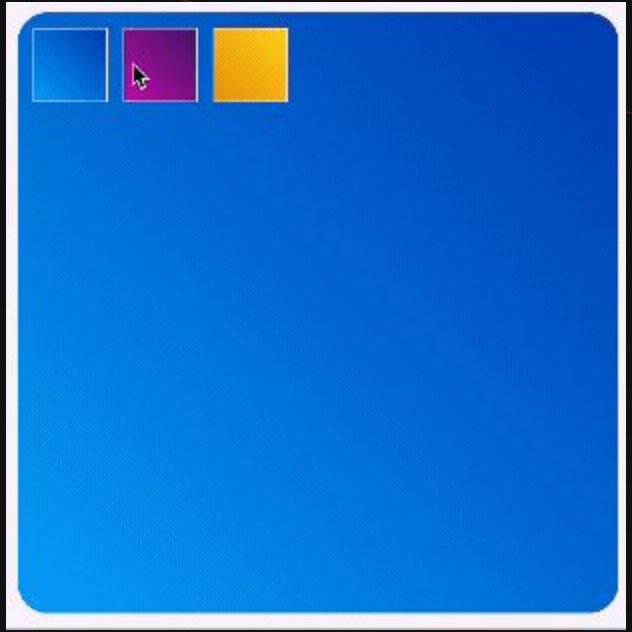
Анимации

surf_tech

```
const CustomPainter({Listenable? repaint}) : _repaint = repaint;
```

```
final Animation<double> animation;
AnimatedPainter(this.animation) : super(repaint: animation);
```

Анимации



Анимации


```
GestureDetector(
  onTapDown: (details) {
    setState(() {
      offset = Offset(
        details.globalPosition.dx,
        details.localPosition.dy,
      );

      _selectedGradient = gradients[i];
    });
  },
```

Анимации


```dart
late final AnimationController _controller;
late final Animation<double> _animation;
_Gradient? _prevColor;
```

Анимации

surf_tech

```dart
@override
void initState() {
  super.initState();
  _controller = AnimationController(
    duration: const Duration(milliseconds: 1000),
    vsync: this,
  );
  _animation = Tween<double>(begin: 0, end: 10).animate(
    CurvedAnimation(
      parent: _controller,
      curve: Curves.easeInOutCirc,
    ),
  );
}
```

Анимации


```dart
@override
Widget build(BuildContext context) {
  return SizedBox.square(
    dimension: _size,
    child: ClipRRect(
      borderRadius: BorderRadius.circular(20),
      child: CustomPaint(
          painter: _SplashPainter(
        offset: widget.offset,
        gradient: widget.gradient,
        prevGradient: _prevColor ?? _Gradient.base(),
        animation: _animation,
      )),
    ),
  );
}
```

Анимации


```
final Animation<double> animation;
final _Gradient gradient;
final _Gradient prevGradient;
final Offset offset;

const _SplashPainter({
  required this.gradient,
  required this.prevGradient,
  required this.offset,
  required this.animation,
}) : super(repaint: animation);
```

Анимации


```
final circleSize = size.width * animation.value;

final radius = circleSize / 2;
final rect = Rect.fromLTWH(0, 0, size.width, size.height);
final circleRect = Rect.fromCircle(center: offset, radius: radius);
```

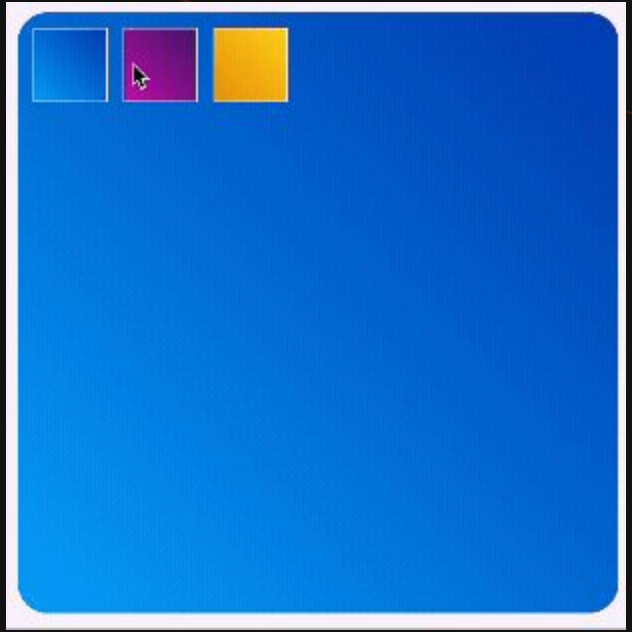
Анимации


```
final paintCurrent = Paint()
  ..shader = ui.Gradient.linear(
    Offset(0, size.height),
    Offset(size.width, 0),
    [
      gradient.start,
      gradient.end,
    ],
  );

final paintPrev = Paint()
  ..shader = ui.Gradient.linear(
    Offset(0, size.height),
    Offset(size.width, 0),
    [
      prevGradient.start,
      prevGradient.end,
    ],
  );
```

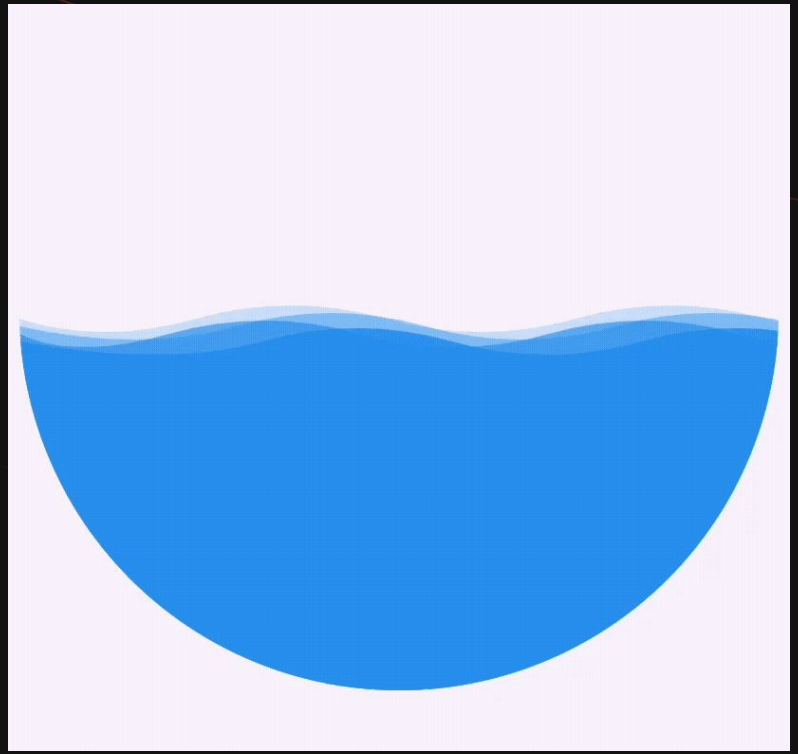
Анимации


```
final transparentCircle = Path.combine(
  PathOperation.difference,
  Path()..addRect(rect),
  Path()
    ..addOval(circleRect)
    ..close(),
);

canvas.drawRect(Rect.largest, paintCurrent);

canvas.drawPath(transparentCircle, paintPrev);
```

Анимации



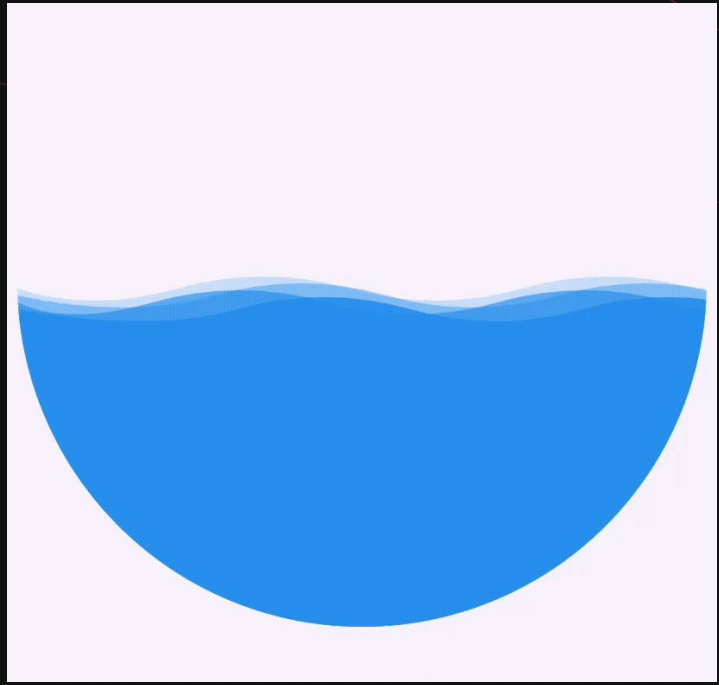
Анимации



Анимации

surf_tech

```dart
@override
void paint(Canvas canvas, Size size) {
  for (var j = 0; j < waveCount; j++) {
    final waveHeigth = size.height * (50 + j) / 100;

    final path = Path()..moveTo(0, waveHeigth);
    for (int i = 0; i < 5; i += 1) {
      final jSign = j.isOdd ? -1 : 1;
      final widthDelta = size.width / 4 * i +
        (jSign * animation.value * size.width / 8 * j / 5);

      final iSign = i.isOdd ? -1 : 1;
      path.quadraticBezierTo(
        widthDelta + size.width / 8,
        waveHeigth + (20 * animation.value) * iSign,
        widthDelta + size.width / 4,
        waveHeigth,
      );
    }

    path
      ..lineTo(size.width, size.height)
      ..lineTo(0, size.height)
      ..close();

    canvas.drawPath(
      path,
      Paint()..color = Colors.blue.withValues(alpha: j / 5),
    );
  }
```



Часть 3

Интерактивность

Интерактивность

surf_tech

Как мы можем реализовать интерактивность?

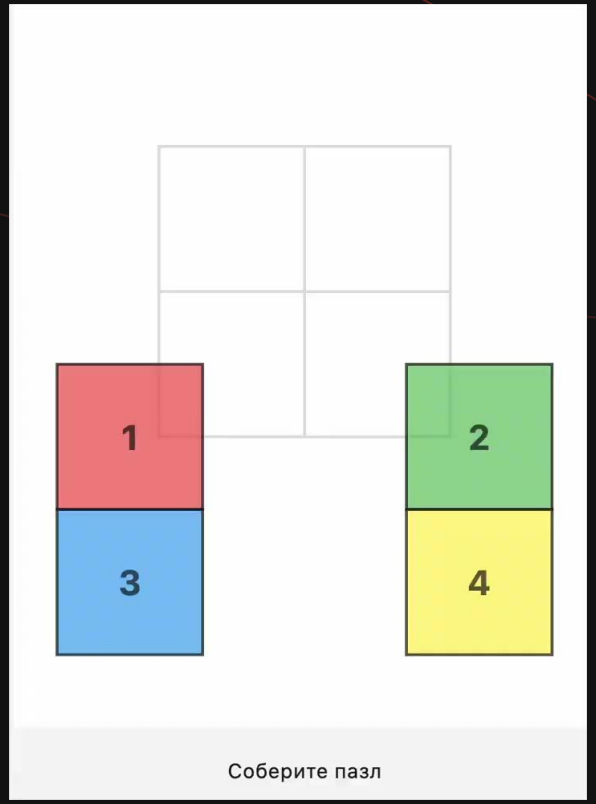
Интерактивность

surf_tech

Как мы можем реализовать интерактивность?

1. **GestureDetector** :

- Разная обработка разных типов жестов



Интерактивность

surf_tech

```dart
class PuzzlePiece {
  final int id;
  final Rect sourceRect;
  final Offset targetPosition;
  final Color color;
  Offset currentPosition;
  bool isPlaced;

  /// Проверка, находится ли точка внутри части
пазла
  bool contains(Offset point, Size pieceSize) {
    final center = currentPosition;
    final rect = Rect.fromCenter(
      center: center,
      width: pieceSize.width,
      height: pieceSize.height,
    );
    return rect.contains(point);
  }
  ....
}
```

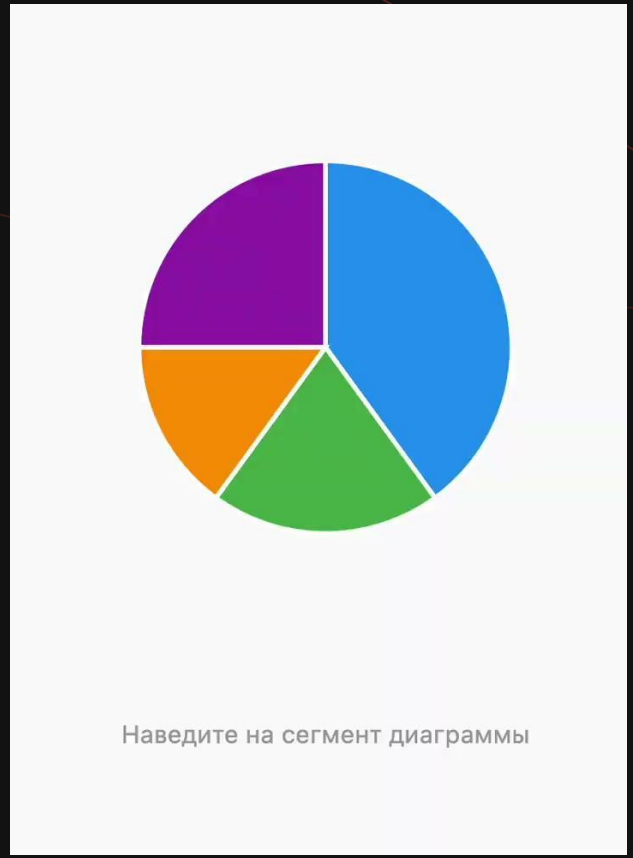
```dart
GestureDetector(
  onScaleStart: _onScaleStart,
  onScaleUpdate: _onScaleUpdate,
  onScaleEnd: _onScaleEnd,
  child: CustomPaint(
    size: const Size(400, 500),
    painter: _PuzzlePainter(
      pieces: puzzlePieces, // List<PuzzlePiece>
      selectedPiece: selectedPiece,
    ),
  ),
),
```

Интерактивность

Как мы можем реализовать интерактивность?

bool? hitTest(Offset position):

- *true* - если hit “попал” по нарисованному контенту
- *false* - если нет
- *null* - пропускаем hit дальше



Интерактивность

surf_tech

```
class _PieChartPainter extends CustomPainter {
  final List<ChartSegment> segments;
  final ValueListenable<ChartSegment?> hoveredSegmentNotifier;
  final double chartRadius;
  final Function(ChartSegment?)? onSegmentSelected;

  // Мутабельное поле для хранения paths сегментов
  final List<Path> _chartPaths = [];

  _PieChartPainter({
    required this.segments,
    required this.hoveredSegmentNotifier,
    required this.chartRadius,
    this.onSegmentSelected,
  }) : super(repaint: hoveredSegmentNotifier);
```

```
  /// Создает Path для сегмента круговой диаграммы
  Path _createSegmentPath(
    Offset center,
    double radius,
    double startAngle,
    double sweepAngle,
  ) {
    final path = Path();
    // Начинаем с центра круга
    path.moveTo(center.dx, center.dy);
    // Создаем прямоугольник для дуги
    final rect = Rect.fromCircle(center: center, radius:
radius);
    // Добавляем дугу от startAngle на sweepAngle радиан
    path.arcTo(rect, startAngle, sweepAngle, false);
    // Закрываем путь (линия от конца дуги к центру)
    path.close();
    return path;
  }
```

Интерактивность

surf_tech

```dart
@override
bool? hitTest(Offset position) {
  // Проверяем попадание используя сохраненные paths
  if (_chartPaths.isEmpty) {
    // Если paths еще не созданы, возвращаем null
    return null;
  }

  // Проверяем каждый path
  for (int i = 0; i < _chartPaths.length; i++) {
    final path = _chartPaths[i];

    // Используем path.contains() для точной проверки попадания
    if (path.contains(position)) {
      final segment = segments[i];

      // Обновляем ValueNotifier (автоматически триггерит перерисовку!)
      if (hoveredSegmentNotifier.value != segment) {
        // Опциональный callback для widget-level логики
        onSegmentSelected?.call(segment);
      }

      return true; // Получать события для этой области
    }
  }
  // Не попали ни в один сегмент - сбрасываем выделение
  onSegmentSelected?.call(null);
  return true;
}
```

Часть 3

Перформанс

Оптимизация

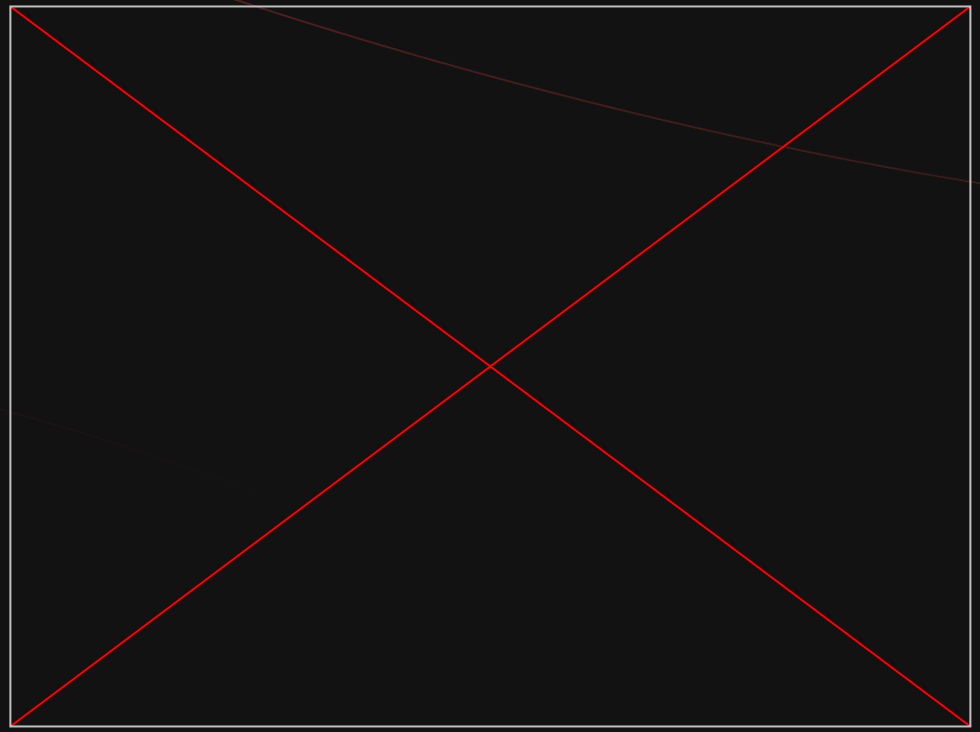
surf_tech

- drawVertices / drawVerticesRaw
- drawPoints / drawPointsRaw
- canvas.saveLayer
- Picture/PictureRecorder

surf_tech

DrawVetices

DrawVertices



Vertices

surf_tech

Vertices - это набор точек, которые объединяются в треугольники.

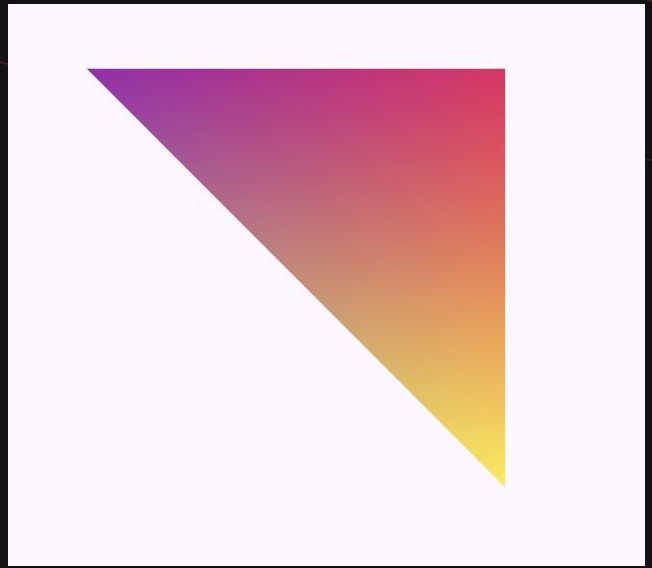
К каждой точке может быть присвоен цвет.

- VertexMode
- positions
- Colors
- Indeces

Vertices

surf_tech

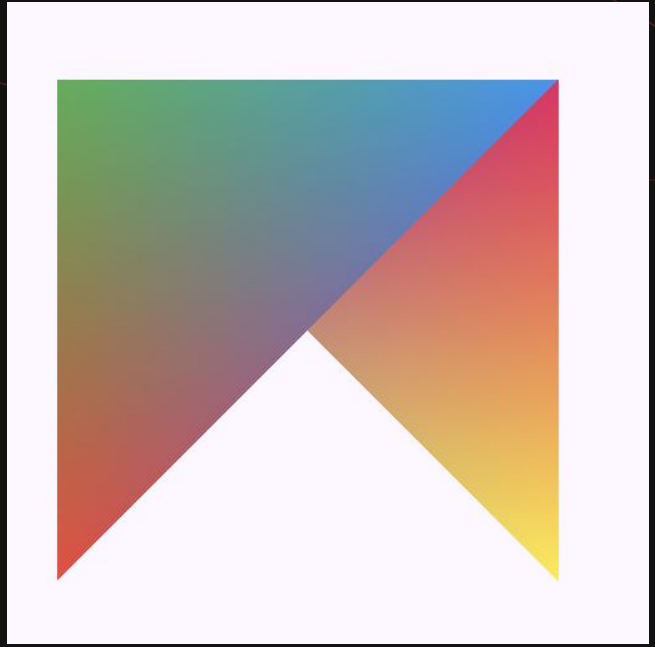
```
final vertices = Vertices(
  VertexMode.triangles,
  [
    center + const Offset(100, 100),
    center + const Offset(100, -100),
    center + const Offset(-100, -100),
  ],
  colors: [
    Colors.yellow,
    Colors.pink,
    Colors.purple,
  ],
);

canvas.drawVertices(vertices, BlendMode.srcOver, paint);
```



Vertices

surf_tech

```dart
final vertices = Vertices(
  VertexMode.triangles,
  [
    center + const Offset(100, 100),
    center + const Offset(100, -100),
    center + const Offset(-100, -100),
    center + const Offset(-100, 100),
    center + const Offset(-100, -100),
    center + const Offset(100, -100),
  ],
  colors: [
    Colors.yellow,
    Colors.pink,
    Colors.purple,
    Colors.red,
    Colors.green,
    Colors.blue,
  ],
);
```



Vertices

surf_tech

```
final vertices = Vertices(
  VertexMode.triangles,
  [
    center + const Offset(100, 100),
    center + const Offset(100, -100),
    center + const Offset(-100, -100),
    center + const Offset(-100, 100),
    center + const Offset(-100, -100),
    center + const Offset(100, -100),
  ],
  colors: [
    Colors.yellow,
    Colors.pink,
    Colors.purple,
    Colors.red,
    Colors.green,
    Colors.blue,
  ],
  indices: [0, 1, 2, 0, 2, 3],
);
```

Vertices.raw

surf_tech

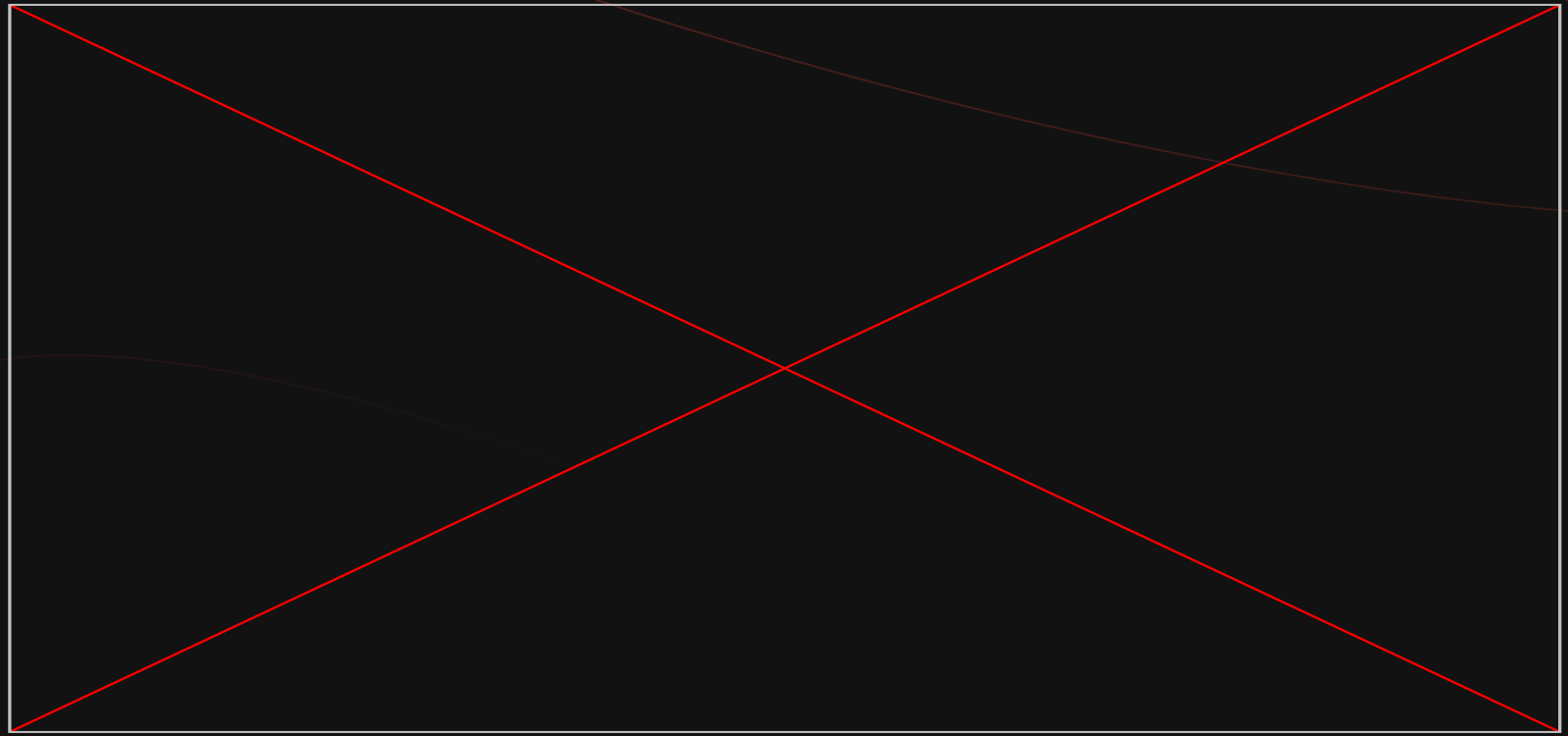
Vertices на typed data.

Vertices.raw

surf_tech

```dart
final Float32List positions = Float32List.fromList([
  center.dx + 100, center.dy + 100, // 0
  center.dx + 100, center.dy - 100, // 1
  center.dx - 100, center.dy - 100, // 2
  center.dx - 100, center.dy + 100, // 3
  center.dx - 100, center.dy - 100, // 4
  center.dx + 100, center.dy - 100, // 5
]);

final Int32List colors = Int32List.fromList([
  Colors.yellow.toARGB32(),
  Colors.pink.toARGB32(),
  Colors.purple.toARGB32(),
  Colors.red.toARGB32(),
  Colors.green.toARGB32(),
  Colors.blue.toARGB32(),
]);

final Uint16List indices = Uint16List.fromList([
  0, 1, 2, // первый треугольник
  0, 2, 3, // второй треугольник
]);

final verticesRaw = Vertices.raw(
  VertexMode.triangles,
  positions,
  colors: colors,
  indices: indices,
);

canvas.drawVertices(verticesRaw, BlendMode.srcOver, paint);
```

Vertices.raw



Draw points

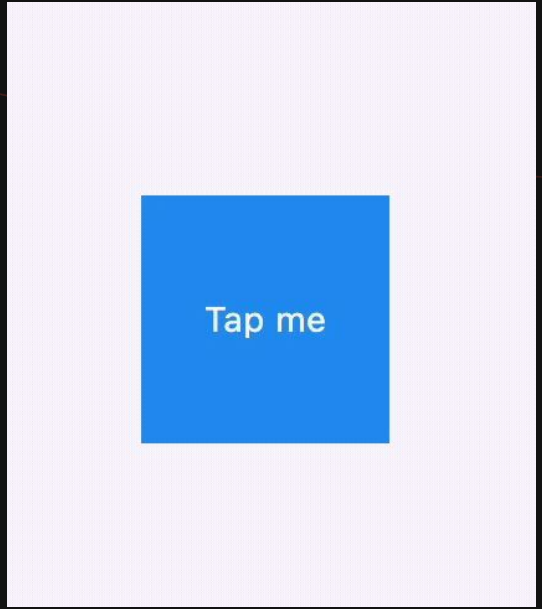
Еще один метод отрисовки набора объектов:

- **PointMode**:
  - points - нарисовать каждую точку отдельно
  - lined - соединить парные точки
  - polygon - соединить всен точки в многоугольник
- Paint:
  - strokeWidth - размер точек
  - strokeCap - фора точек

Draw points

surf_tech

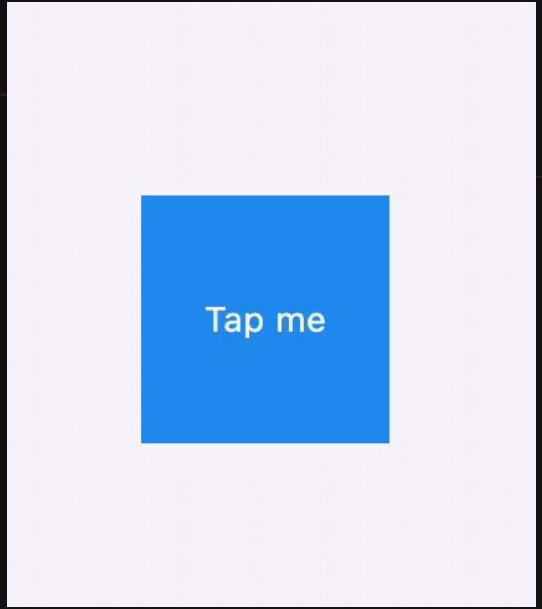
```
void paint(Canvas canvas, Size size) {
  final center = size.center(Offset.zero);
  final progress = Curves.easeOut.transform(animation.value);

  final list = <Offset>[];
  for (final p in particles) {
    final dx = cos(p.angle) * p.distance * progress;
    final dy = sin(p.angle) * p.distance * progress;
    final particleCenter = center + Offset(dx, dy);

    list.add(particleCenter);
  }
  final paint = Paint()
    ..color = Colors.blue
    ..strokeWidth = 5
    ..strokeCap = StrokeCap.round;
  canvas.drawPoints(PointMode.points, list, paint);
}
```



Draw raw points

surf_tech

```dart
final center = size.center(Offset.zero);
final progress = Curves.easeOut.transform(animation.value);

final points = Float32List(particles.length * 2);
for (int i = 0; i < particles.length; i++) {
  final p = particles[i];
  final dx = cos(p.angle) * p.distance * progress;
  final dy = sin(p.angle) * p.distance * progress;
  final particleCenter = center + Offset(dx, dy);

  points[i * 2] = particleCenter.dx;
  points[i * 2 + 1] = particleCenter.dy;
}

final paint = Paint()
  ..color = Colors.blue
  ..strokeWidth = 5
  ..strokeCap = StrokeCap.round;
canvas.drawRawPoints(PointMode.points, points, paint);
```



surf_tech

Canvas.saveLayer

saveLayer

1. Создает новый layer определенного размера в стеке рендеринга Canvas;
2. Все операции рисования, выполненные после вызова, происходят в отдельном буфере
3. Этот буфер применяется к основному canvas с учетом указанных параметров paint

Параметры:

- *Rect? rect* - область отсечения. Если `null`, слой применяется ко всему canvas;
- Paint paint - кисть, определяющая как слой будет композитирован (прозрачность, blend mode, фильтры).

saveLayer vs save

surf_tech

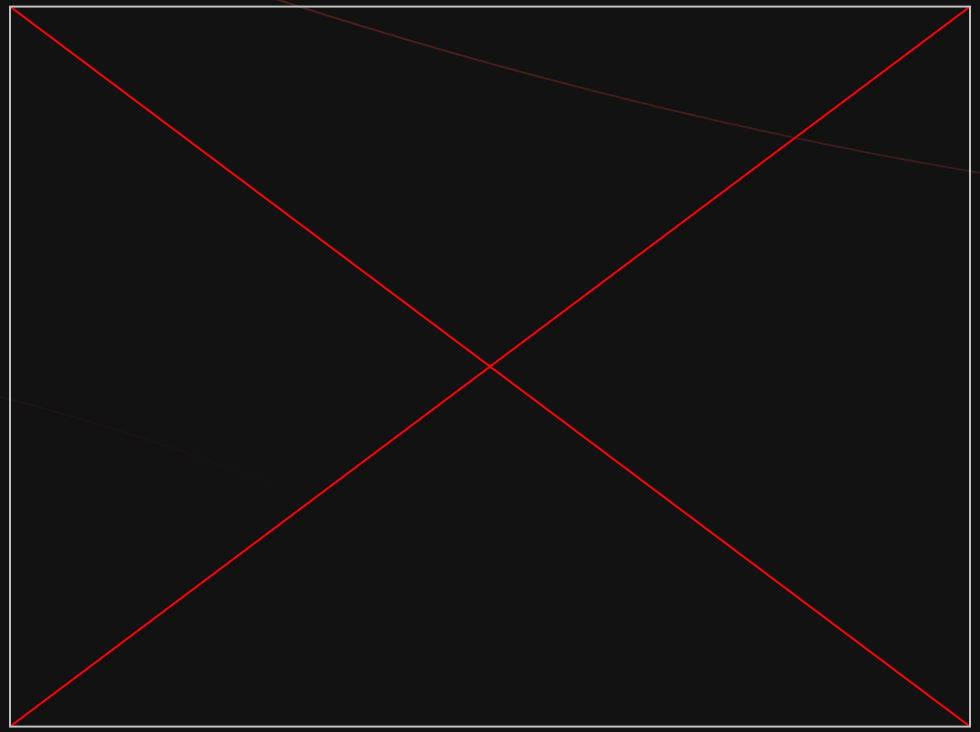
Save:

- Сохраняет только трансформации (translate, rotate, scale, clipRect);
- Стоит дешево;
- НЕ создает новый буфер.

SaveLayer:

- Новый layer в рамках GPU, ко всем объектам которого применяется paint эффекты;
- Стоит дороже;
- Создает отдельный буфер рендеринга;

saveLayer



saveLayer


```dart
/// SaveLayer создает отдельный буфер, где все элементы рисуются
/// с полной непрозрачностью, а затем весь буфер целиком
/// применяется к канвасу с указанной прозрачностью
void _drawWithSaveLayer(Canvas canvas, Offset center) {
  // Создаем Paint с прозрачностью для всего слоя
  final layerPaint = Paint()
    ..color = Colors.black.withValues(
      alpha: 0.6,
    );

  canvas.saveLayer(null, layerPaint);
  // Рисуем группу кругов
  _drawCircleGroup(canvas, center);
  // Restore применяет слой с указанной прозрачностью
  canvas.restore();
}
```

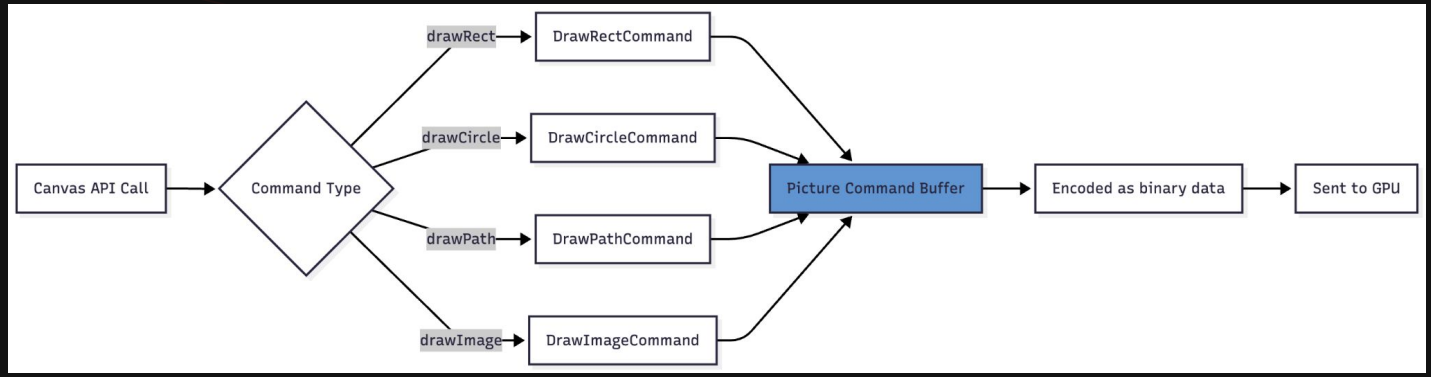
Picture

Это не растровое изображение, это набор графических инструкций.

Для чего используется:

- Кеширование
- Оптимизация работы графикой

Используется вместе с **PictureRecorder**.



Picture


```
ui.Picture? _picture;

FixedStarPainter();

ui.Picture _drawStar(Size size) {
  final recorder = ui.PictureRecorder();
  final canvas = Canvas(recorder);

  /// ... рисуем звезу

  canvas.drawPath(path, paintFill);
  canvas.drawPath(path, paintStroke);

  return recorder.endRecording();
}
```

Picture

surf_tech

```dart
@override
void paint(Canvas canvas, Size size) {
  _picture ??= _drawStar(size);

  if (_picture case final pic?) {
    canvas.drawPicture(pic);
  }
}
```

Часть 6

LAYERING

surf_tech

Custom painter - это фундамент Flutter

RenderObject

surf_tech

```dart
// RENDER PARAGRAPH

@override
  void paint(PaintingContext context, Offset offset) {
    _layoutTextWithConstraints(constraints);
    assert(() {
      if (debugRepaintTextRainbowEnabled) {
        final Paint paint = Paint()..color =
debugCurrentRepaintColor.toColor();
        context.canvas.drawRect(offset & size, paint);
      }
      return true;
    }());

    if (_needsClipping) {
      final Rect bounds = offset & size;
      if (_overflowShader != null) {
        context.canvas.saveLayer(bounds, Paint());
      } else {
        context.canvas.save();
      }
      context.canvas.clipRect(bounds);
    }

    if (_lastSelectableFragments != null) {
      for (final _SelectableFragment fragment in
_lastSelectableFragments!) {
        fragment.paint(context, offset);
      }
    }

    _textPainter.paint(context.canvas, offset);

    paintInlineChildren(context, offset);

    if (_needsClipping) {
      if (_overflowShader != null) {
        context.canvas.translate(offset.dx, offset.dy);
        final Paint paint =
            Paint()
              ..blendMode = BlendMode.modulate
              ..shader = _overflowShader;
        context.canvas.drawRect(Offset.zero & size, paint);
      }
      context.canvas.restore();
    }
  }
```

RenderObject

surf_tech

```dart
/// _RenderColoredBox
@override
  void paint(PaintingContext context, Offset offset) {
    if (size > Size.zero) {
      context.canvas.drawRect(offset & size, Paint()..color = color);
    }
    if (child != null) {
      context.paintChild(child!, offset);
    }
  }
```

PaintingContext

surf_tech

```
class PaintingContext extends ClipContext {
  final ContainerLayer _containerLayer;
  PictureLayer? _currentLayer;
  ui.PictureRecorder? _recorder;
  Canvas? _canvas;
}
```

PaintingContext

Это ключевой объект в архитектуре рендеринга Flutter, который:

- управляет процессом создания Picture и отрисовкой;
- формирования LayerTree во время paint-фазы;

PaintingContext

Это ключевой объект в архитектуре рендеринга Flutter, который:

- управляет процессом создания Picture и отрисовкой;
- формирования LayerTree во время paint-фазы;

При использовать CustomPainter *canvas* отдается PaintingContext.

Custom Paint

surf_tech

```dart
@override
void paint(PaintingContext context, Offset offset) {
  if (_painter != null) {
    _paintWithPainter(context.canvas, offset, _painter!);
    _setRasterCacheHints(context);
  }
  super.paint(context, offset);
  if (_foregroundPainter != null) {
    _paintWithPainter(context.canvas, offset, _foregroundPainter!);
    _setRasterCacheHints(context);
  }
}
```

PainingContext

Это ключевой объект в архитектуре рендеринга Flutter, который:

- управляет процессом создания Picture и отрисовкой;
- формирования LayerTree во время paint-фазы;

При использовать CustomPainter *canvas* отдается PaintingContext.

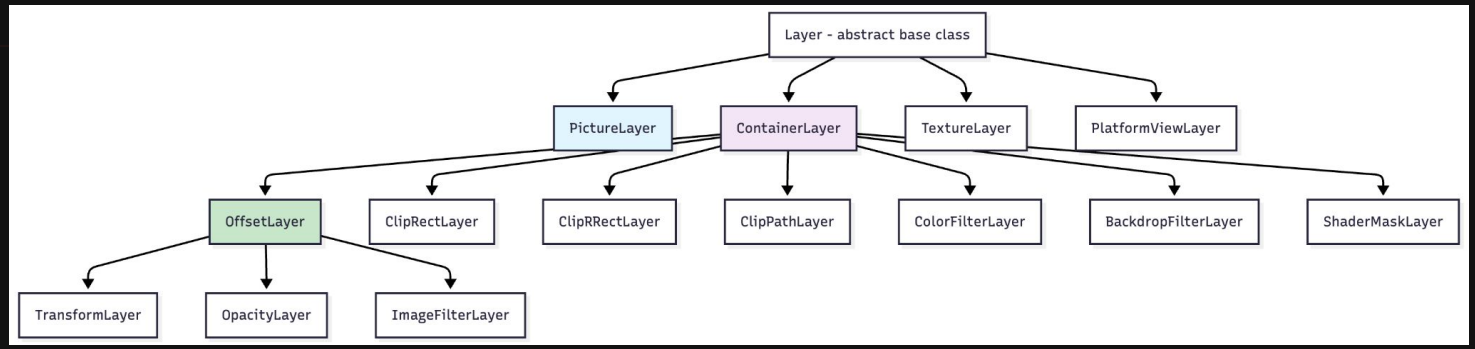
Один layer = один PainingContext

Layer

Это композитный узел в **LayerTree.**

Layer оборачивает Picture и добавляет дополнительные свойства.

Свойства зависят от типа слоя:



Типы слоев

1. ***PictureLayer*** - тип слоя, которые содержит Picture. Единственный слой, который работает с графикой
2. ***ContainerLayer*** - слой, группирующий другие слои
3. ***TransformLayer*** - слой, применяющий трансформацию Matrix4
4. ***OpacityLayer*** - слой, применяющий прозрачность
5. ***ClipRectLayer*** - слой с обрезкой
6. ***TextureLayer*** - слой, накладывающий текстуру на бекграунд

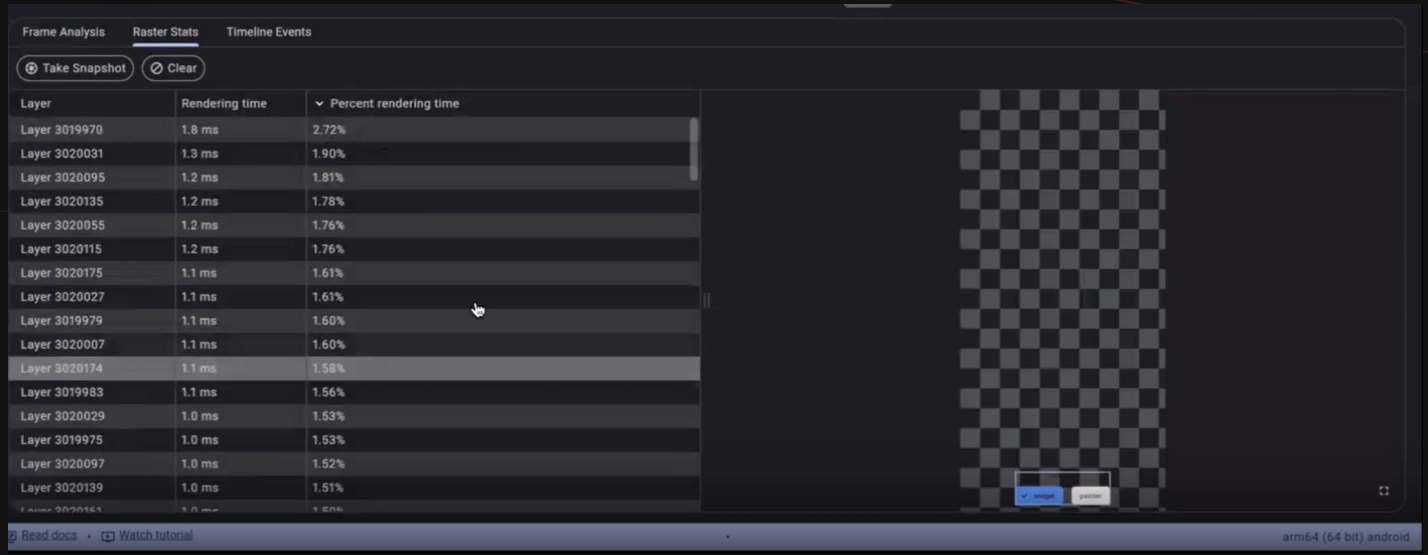
*Не для каждого виджет создается отдельный Layer

Специальные Layer создают только виджеты с особыми свойствами (Opacity, Transform, ClipRRect) или с *isRepaintBoundary == true*

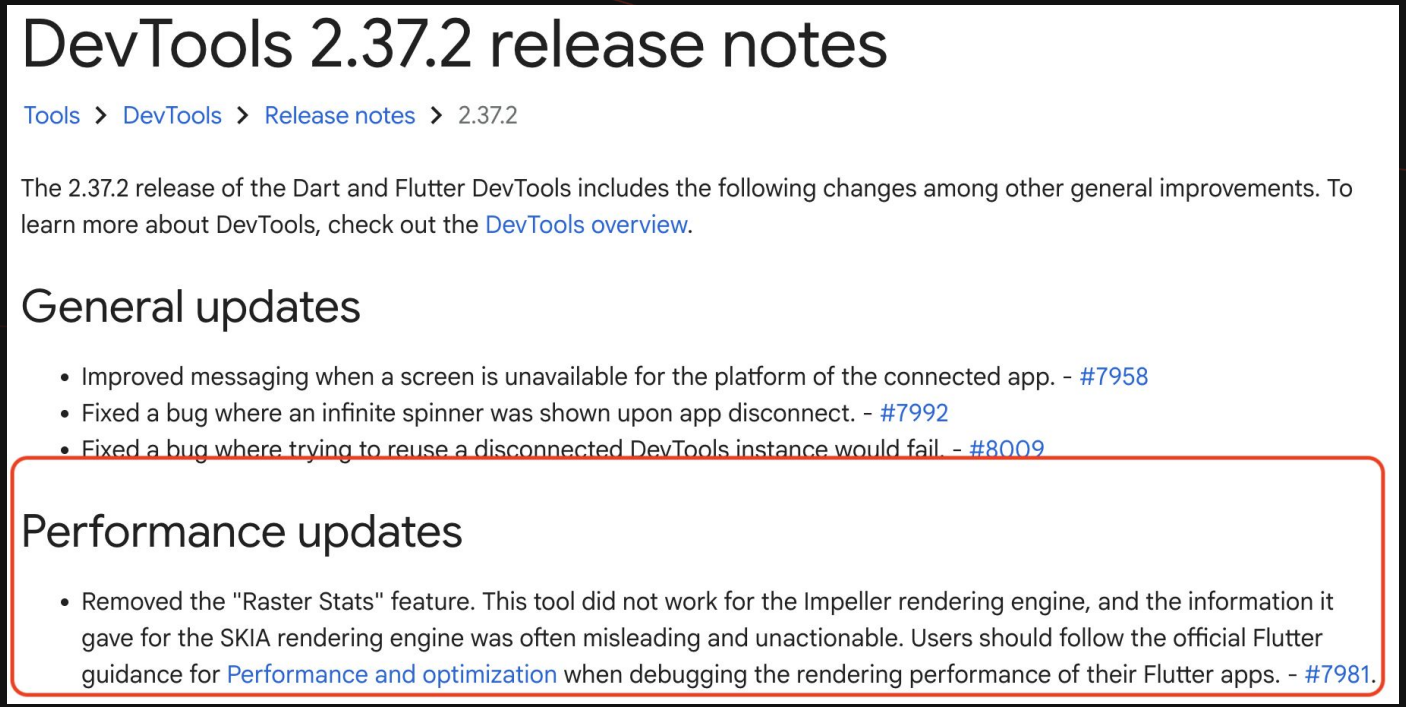
Layer

surf_tech

Можно было посмотреть в RasterStats в DevTools



Layer



DevTools 2.37.2 release notes

Tools > DevTools > Release notes > 2.37.2

The 2.37.2 release of the Dart and Flutter DevTools includes the following changes among other general improvements. To learn more about DevTools, check out the DevTools overview.

General updates

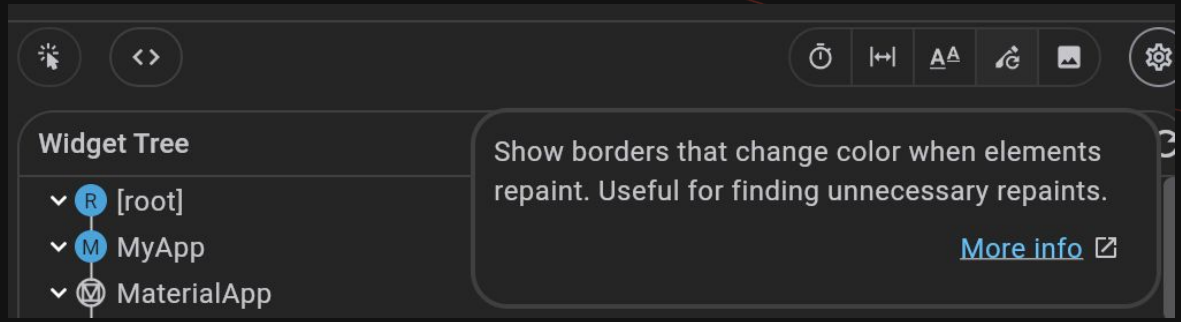
- Improved messaging when a screen is unavailable for the platform of the connected app. - #7958
- Fixed a bug where an infinite spinner was shown upon app disconnect. - #7992
- Fixed a bug where trying to reuse a disconnected DevTools instance would fail. - #8009

Performance updates

- Removed the "Raster Stats" feature. This tool did not work for the Impeller rendering engine, and the information it gave for the SKIA rendering engine was often misleading and unactionable. Users should follow the official Flutter guidance for Performance and optimization when debugging the rendering performance of their Flutter apps. - #7981.

ИСТОЧНИК

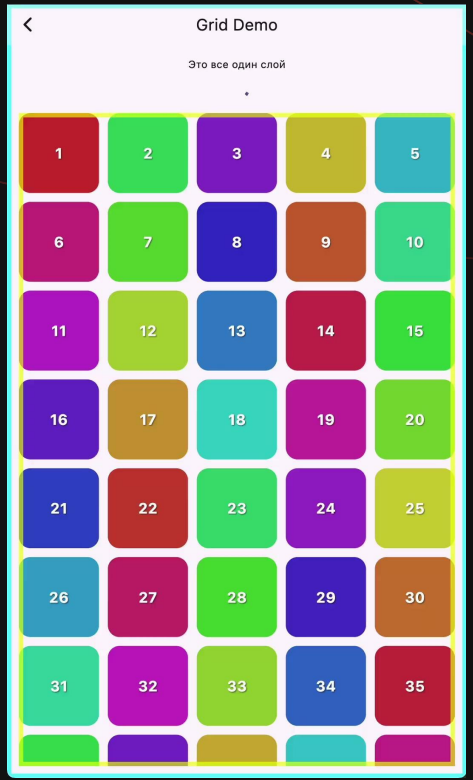
Layer


Widget Tree
[root]
MyApp
MaterialApp
Show borders that change color when elements repaint. Useful for finding unnecessary repaints.
More info

Layer

surf_tech

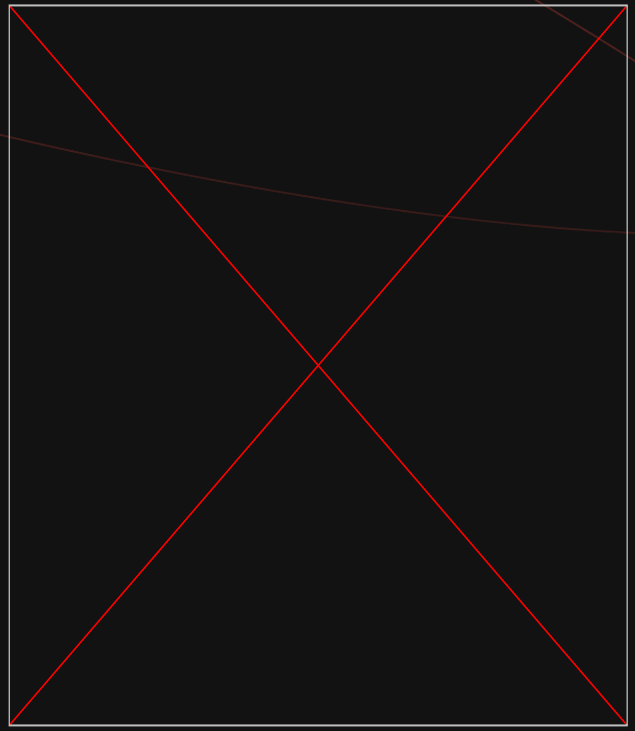
```
GridView.builder(
  addRepaintBoundaries: false,
  gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 5,
    crossAxisSpacing: 12,
    mainAxisSpacing: 12,
    childAspectRatio: 1.0,
  ),
  itemCount: 50,
  itemBuilder: (context, index) {
    return HoverGridItem(
      index: index,
      baseColor: _getColorForIndex(index),
    );
  },
),
```



Layer

surf_tech

```
GridView.builder(
   gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
     crossAxisCount: 5,
     crossAxisSpacing: 12,
     mainAxisSpacing: 12,
     childAspectRatio: 1.0,
   ),
   itemCount: 50,
   itemBuilder: (context, index) {
     return RepaintBoundary(
       child: HoverGridItem(
         index: index,
         baseColor: _getColorForIndex(index),
       ),
     );
   },
 ),
```



Layer

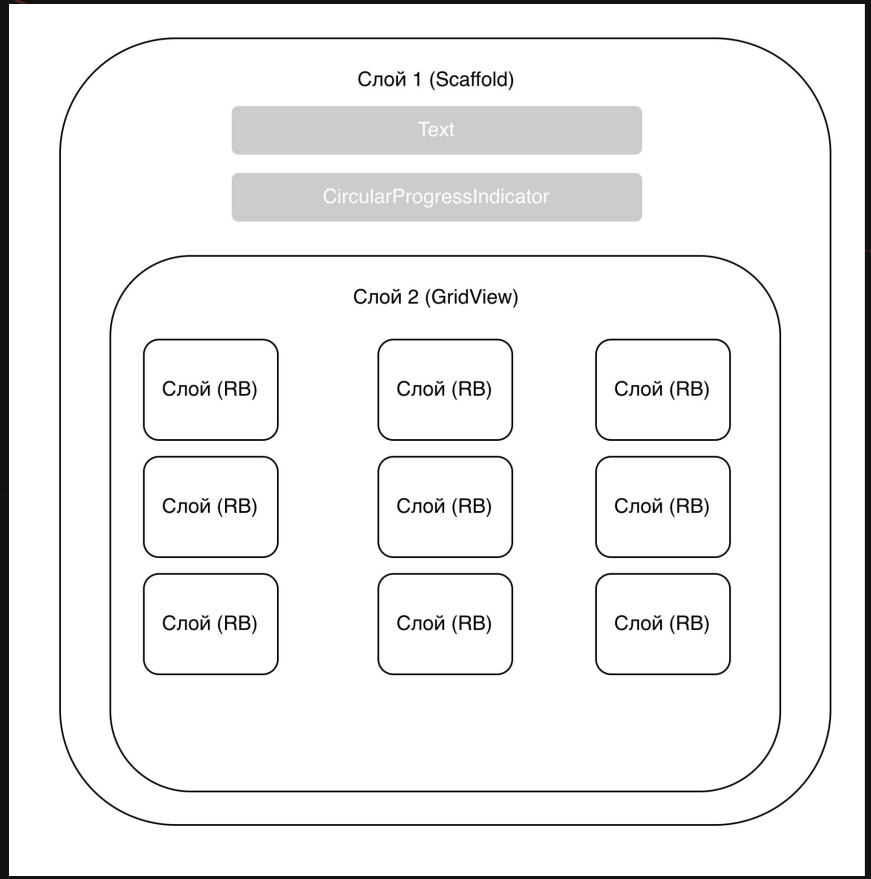

```dart
class RenderRepaintBoundary extends RenderProxyBox {
  /// Creates a repaint boundary around [child].
  RenderRepaintBoundary({RenderBox? child}) : super(child);

  @override
  bool get isRepaintBoundary => true;
}
```

Layer

surf_tech

```
  static void _repaintCompositedChild(
    RenderObject child, {
    bool debugAlsoPaintedParent = false,
    PaintingContext? childContext,
  }) {

    OffsetLayer? childLayer = child._layerHandle.layer as OffsetLayer?;
    if (childLayer == null) {
      final OffsetLayer layer = child.updateCompositedLayer(oldLayer: null);
      child._layerHandle.layer = childLayer = layer;
    } else {
      Offset? debugOldOffset;
      childLayer.removeAllChildren();
      final OffsetLayer updatedLayer = child.updateCompositedLayer(oldLayer:
childLayer);
    }
    child._needsCompositedLayerUpdate = false;

    childContext ??= PaintingContext(childLayer, child.paintBounds);
    child._paintWithContext(childContext, Offset.zero);

    childContext.stopRecordingIfNeeded();
  }
```

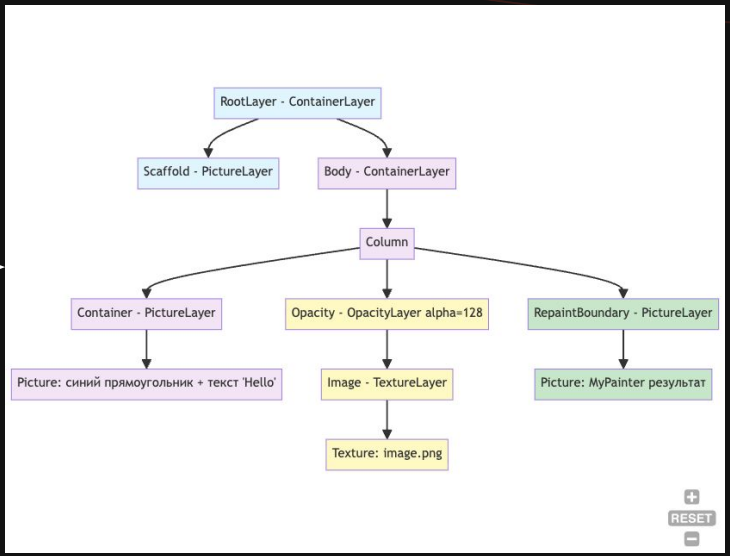
Layer


Слой 1 (Scaffold)
Text
CircularProgressIndicator
Слой 2 (GridView)
Слой (RB)
Слой (RB)
Слой (RB)
Слой (RB)
Слой (RB)
Слой (RB)
Слой (RB)
Слой (RB)
Слой (RB)

LayerTree

Это **иерархическая структура Layer объектов**, собранная во время paint фазы. Это дерево представляет всю графику приложения и строится во время **Paint Phase** рендеринга. Это происходит автоматически для всех виджетов.

```
MaterialApp(
  home: Scaffold(
    body: Column(
      children: [
        Container(
          color: Colors.blue,
          child: Text('Hello'),
        ),
        Opacity(
          opacity: 0.5,
          child: Image.asset('image.png'),
        ),
        RepaintBoundary(
          child: CustomPaint(
            painter: MyPainter(),
          ),
        ),
      ],
    ),
  ),
)
```

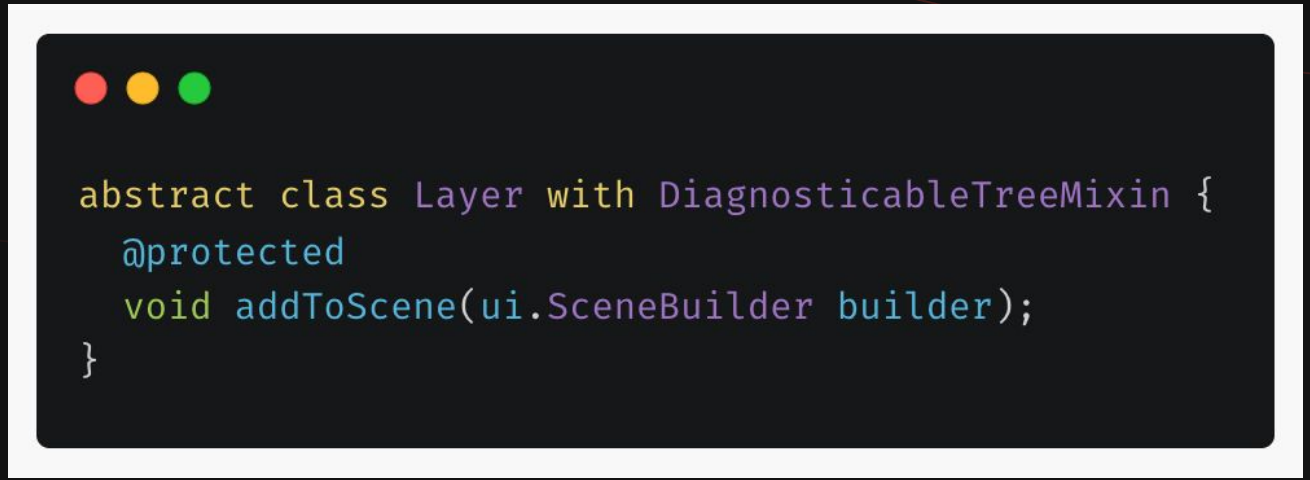


LayerTree


```
class PaintingContext extends ClipContext {
  @protected
  void appendLayer(Layer layer) {
    assert(!_isRecording);
    layer.remove();
    _containerLayer.append(layer);
  }
}
```

LayerTree

surf_tech



```
abstract class Layer with DiagnosticableTreeMixin {
  @protected
  void addToScene(ui.SceneBuilder builder);
}
```

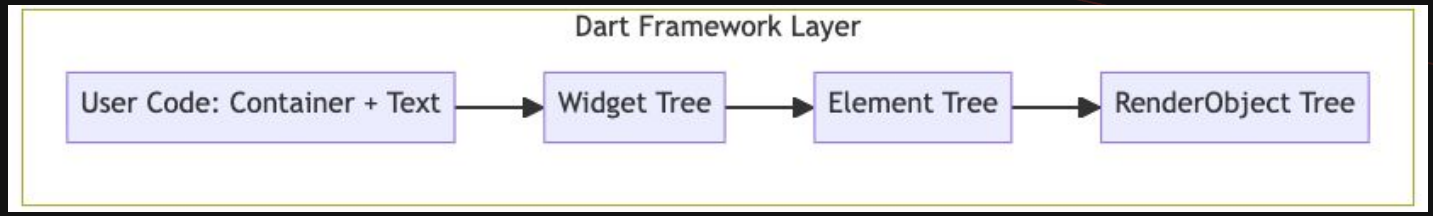
Scene

Это объект, представляющий финальную сцену для отправки в GPU.
Scene создается из LayerTree.

SceneBuilder - это класс для построения Scene из LayerTree.

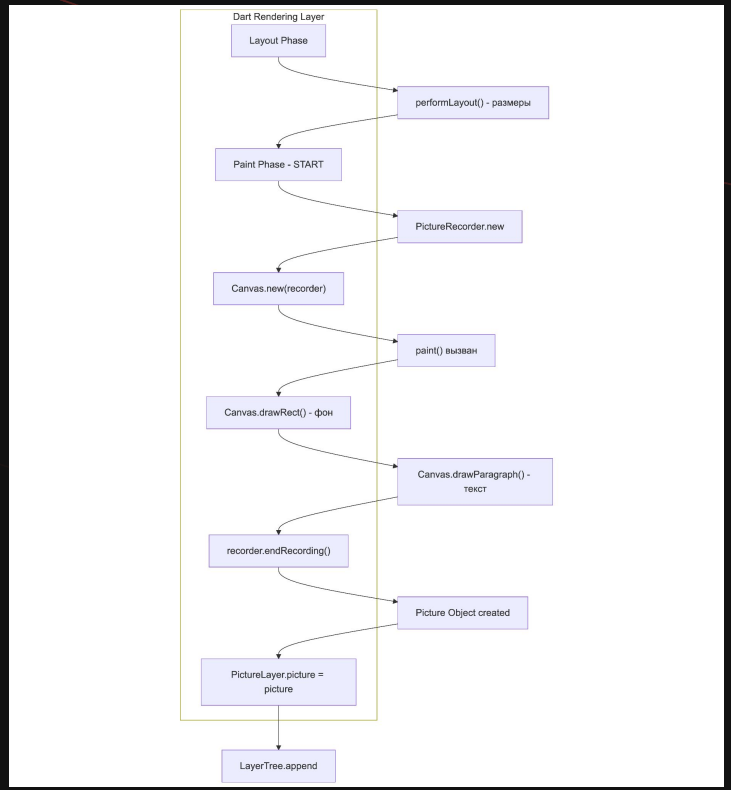
```
abstract class Scene {
  Image toImageSync(int width, int height);
  Future<Image> toImage(int width, int height);
  void dispose();
}

@pragma('vm:entry-point')
base class _NativeScene extends
NativeFieldWrapperClass1 implements Scene {
  /// external вызовы
}
```

Схемка


Dart Framework Layer
User Code: Container + Text
Widget Tree
Element Tree
RenderObject Tree

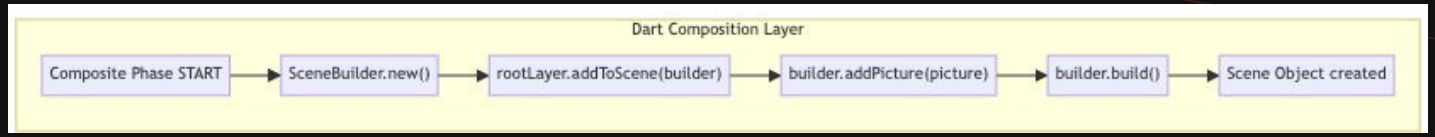
Схемка

surf_tech



Схемка

surf_tech



Итоги

surf_tech

Узнали про:

- Базовове применеие CustomPainter;
- Способы использования;
- Оптимизацию;
- Немного подипдайвили в технологию.

Спасибо за внимание!