

Картинка, которая не влезает в телефон

pan / zoom тяжёлого контента в Compose без потери кадров

**Дарья
Городнянская**

Android-разработчик в Surf

Кто я и о чём я

Кто я и о чём я

СЕЙЧАС



Источник GIF: Tenor

Кто я и о чём я

В ПРОШЛОМ



Источник GIF: Tenor

СЕЙЧАС



Кто я и о чём я

Медтех

[DICOM-срезы · C++]



Анимационная студия

[OpenUSD · 3D-инструменты]



Android-разработка

[Surf · фудтех-проект]



Кто я и о чём я



[04]

Медицинские снимки

Это боль **каждого** разработчика

[04]

Медицинские снимки

Это боль **каждого** разработчика

[01]

Карты

[04]

Медицинские снимки

Это боль **каждого** разработчика

[01]

Карты

[02]

Фоторедакторы

[04]

Медицинские снимки

Это боль каждого разработчика

[01]

Карты

[02]

Фоторедакторы

[03]

**Ленты с тяжёлой
медиа**

[04]

Медицинские снимки

Это боль каждого разработчика

[01]

Карты

[02]

Фоторедакторы

[03]

Ленты с тяжёлой
медиа

[04]

Медицинские снимки

Сделаешь в лоб — тормозит, дёргается
или вообще вылетает.



Как засунуть слона в холодильник?



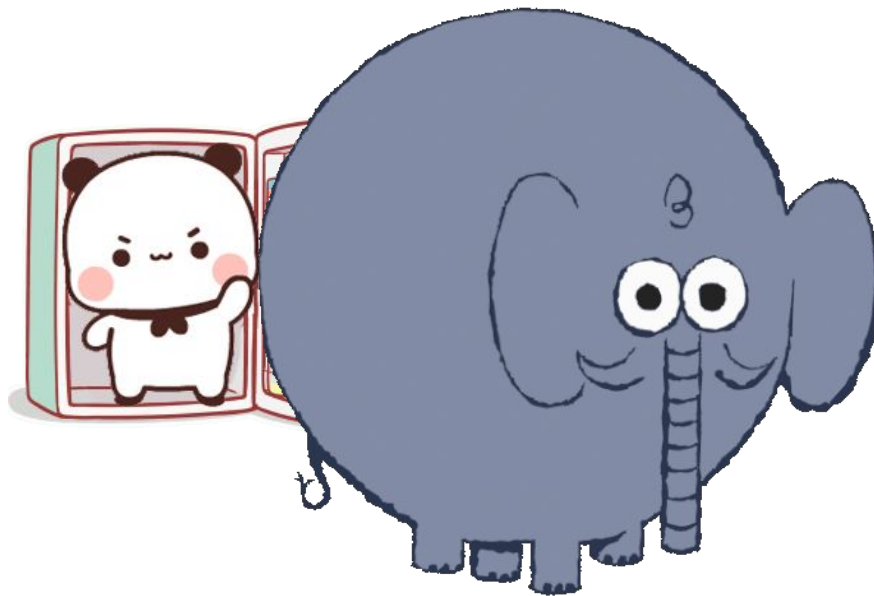
Дано

слон, холодильник



Задача

поместить слона
в холодильник



Как засунуть слона в холодильник?



Дано

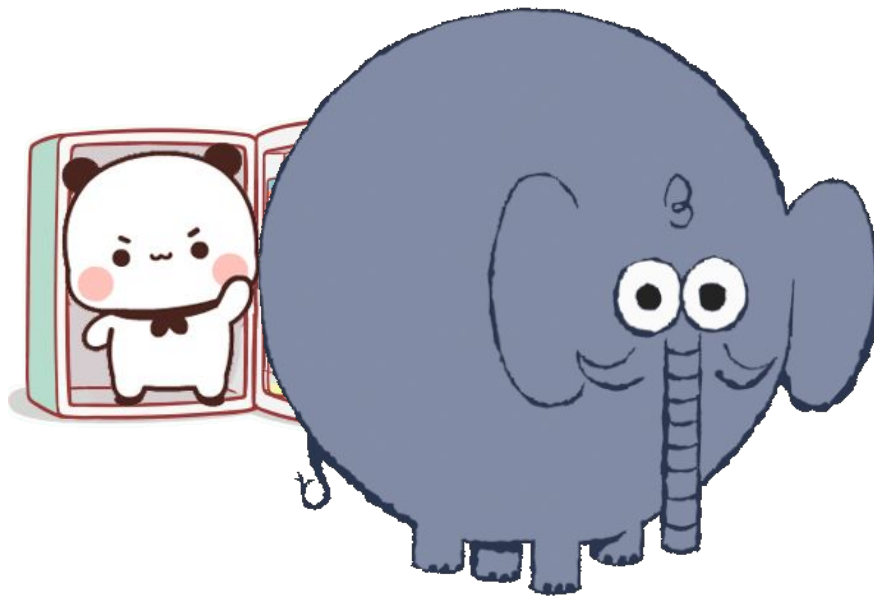
слон, холодильник



Задача

поместить слона
в холодильник

Засовываем только хобот



Как засунуть картинку в телефон?



Дано

огромное изображение,
телефон



Задача

отобразить изображение
на экране

Как засунуть картинку в телефон?



Дано

огромное изображение,
телефон



Задача

отобразить изображение
на экране



Как засунуть картинку в телефон?



Дано

огромное изображение,
телефон



Задача

отобразить изображение
на экране



Как засунуть картинку в телефон?



Дано

огромное изображение,
телефон



Задача

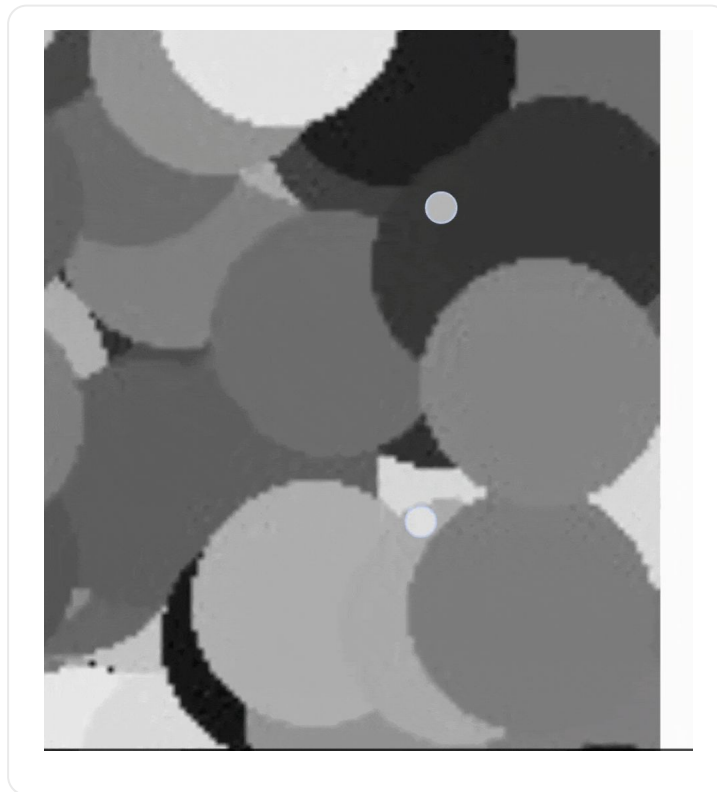
отобразить изображение
на экране

Показываем только фрагмент



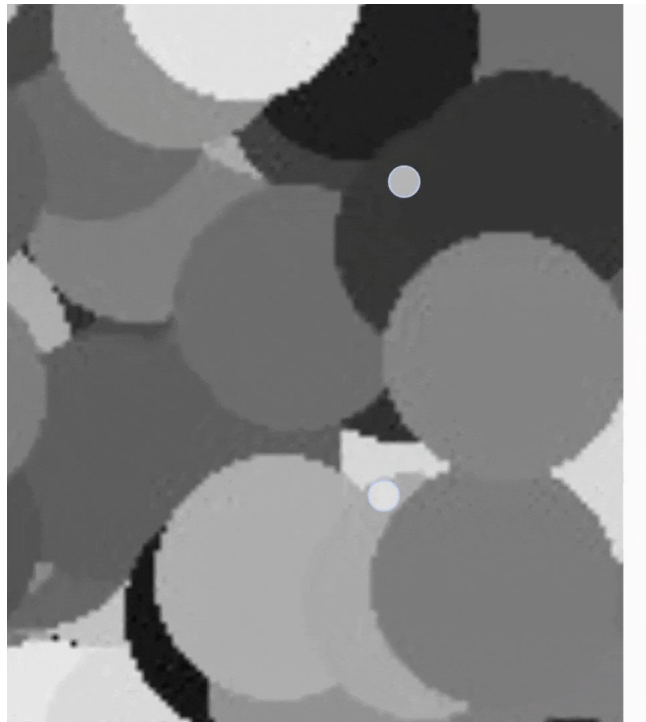
Решение в лоб

Решение в лоб



Решение в лоб

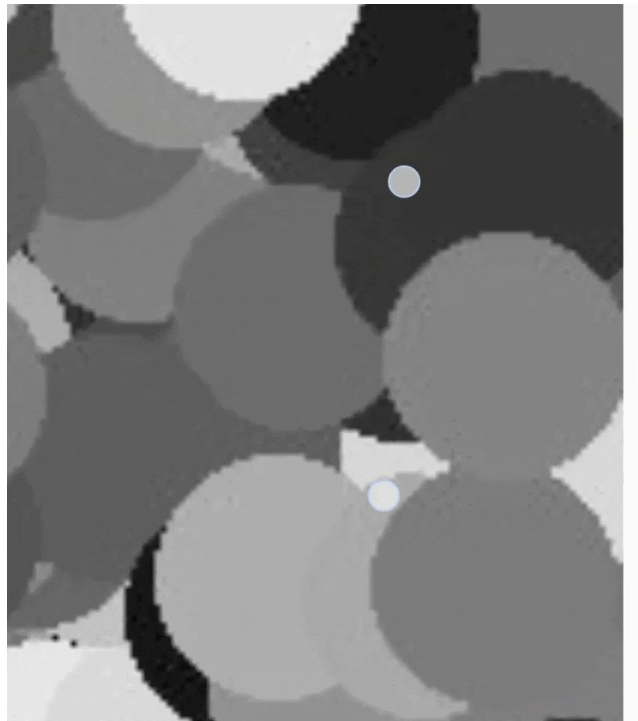
```
withContext(dispatchers.io) {  
    context.assets  
        .open(assetName)  
        .use { BitmapFactory.decodeStream(it) }  
        ?: error("Failed to decode $assetName")  
}
```



Загрузка изображения

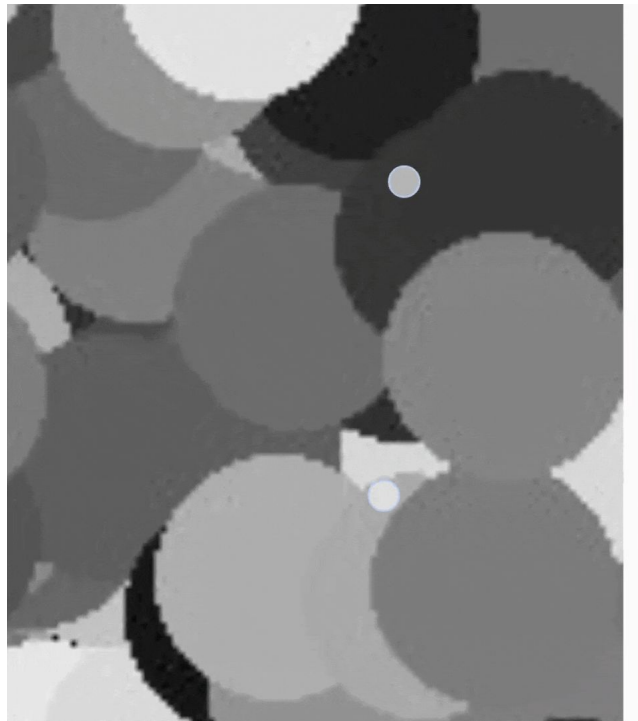
```
val fit = fitScale(
    size.width,
    size.height,
    bitmap.width,
    bitmap.height,
)

drawImage(
    image = bitmap,
    dstOffset = IntOffset.Zero,
    dstSize = IntSize(
        width = (bitmap.width * fit).toInt(),
        height = (bitmap.height * fit).toInt(),
    ),
    filterQuality = FilterQuality.Medium,
)
```



pan/zoom в лоб

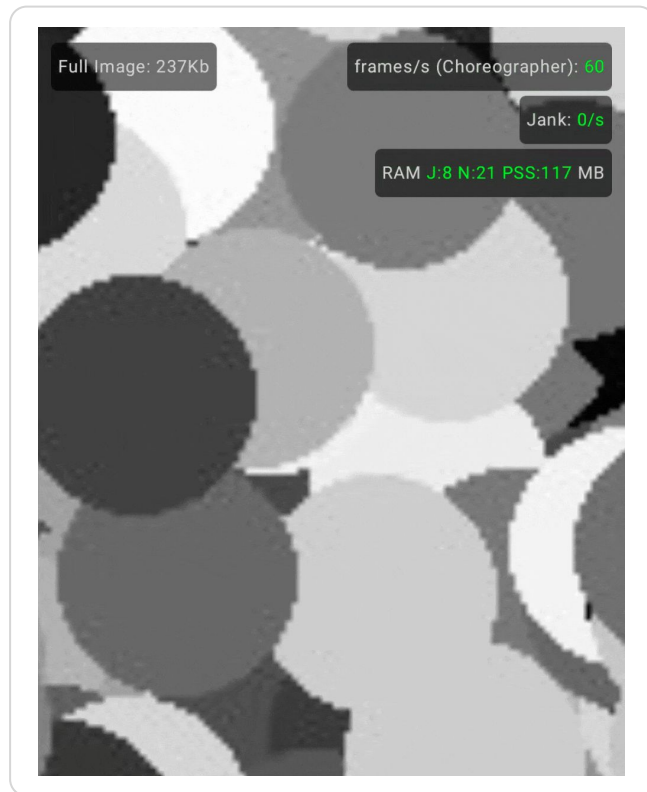
```
Box(  
    modifier  
        .fillMaxSize()  
        .pointerInput(Unit) {  
            detectTransformGestures { centroid, pan,  
zoom, _ -> onGesture(centroid, pan, zoom) }  
        }  
) {  
    Image(  
        bitmap = bitmap.asImageBitmap(),  
        contentDescription = null,  
        modifier = Modifier.graphicsLayer(  
            scaleX = scale,  
            scaleY = scale,  
            translationX = offset.x,  
            translationY = offset.y,  
        )  
    )  
}
```



[Как делать HE надо]

SURF.RU

pan/zoom в лоб

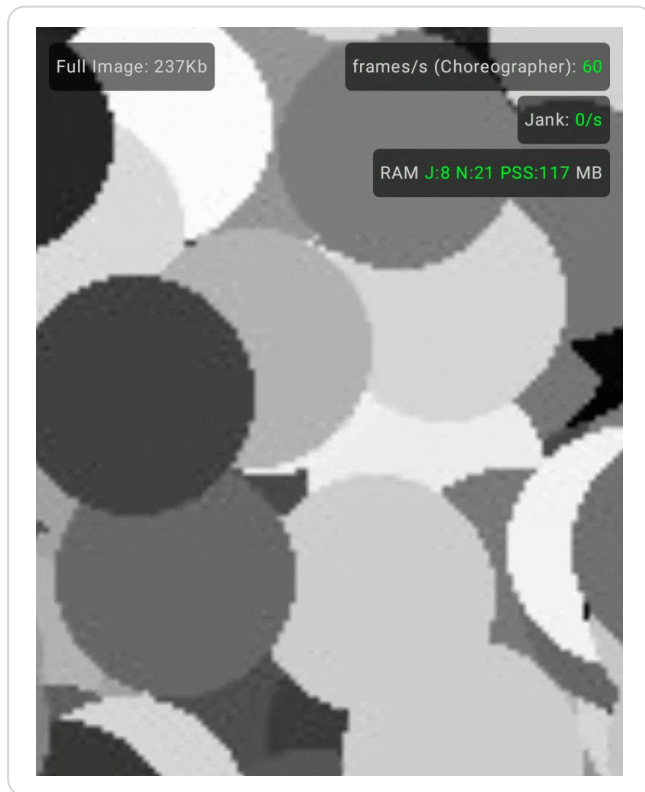


[Как делать HE надо]

SURF.RU

pan/zoom в лоб

test_image_small.jpg



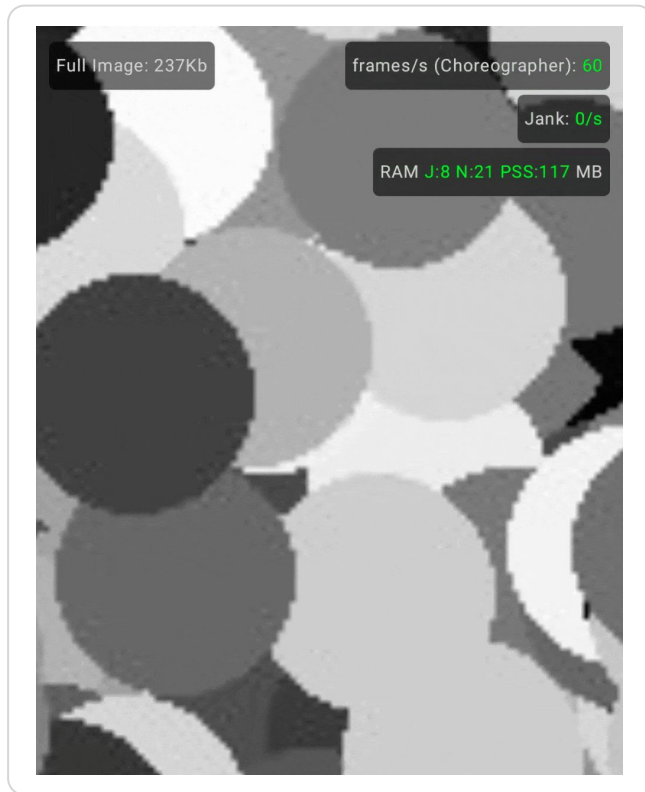
[Как делать HE надо]

SURF.RU

pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

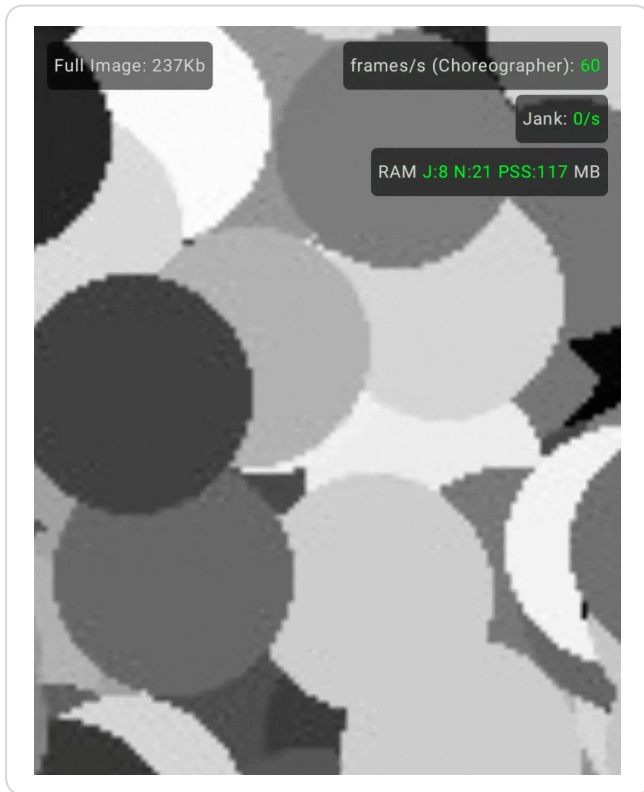


pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

→ 237 Kb



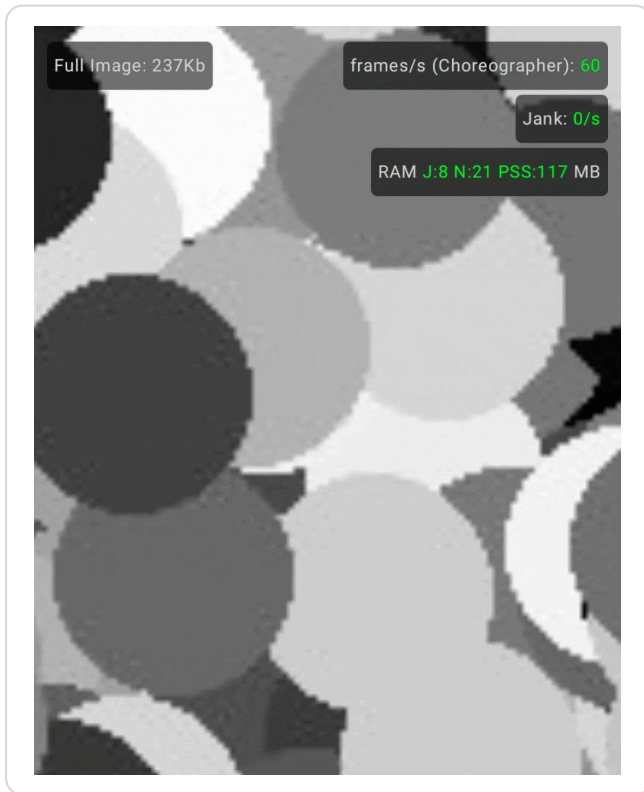
pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

→ 237 Kb

✓ Pan



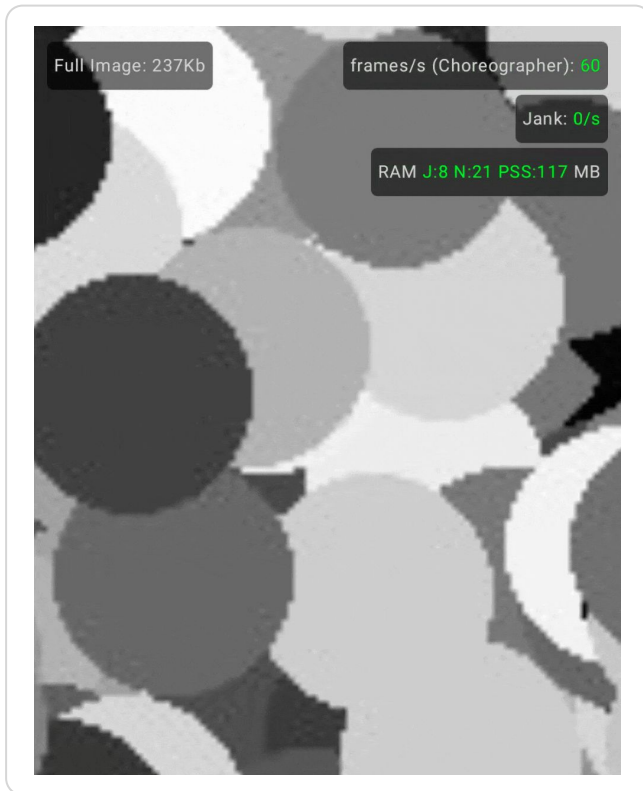
pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

→ 237 Kb

- ✓ Pan
- ✓ Zoom



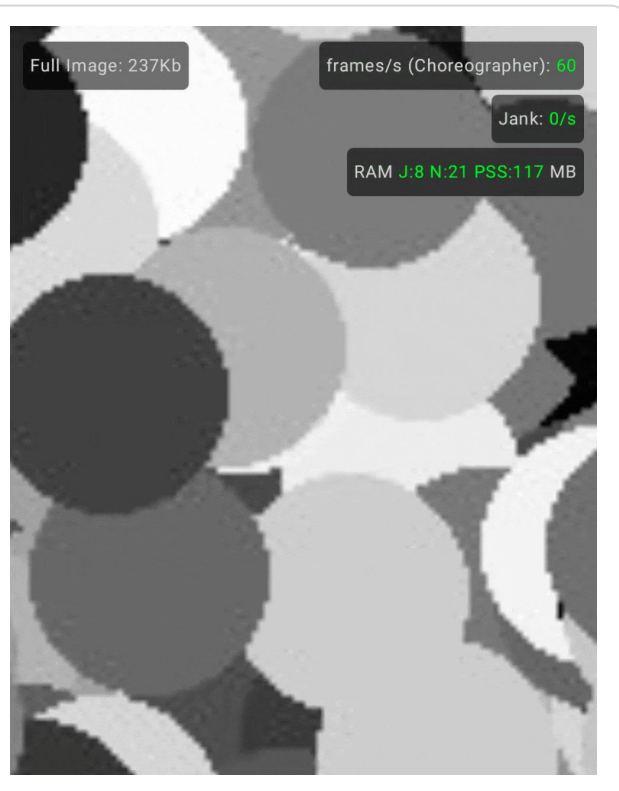
pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

→ 237 Kb

- ✓ Pan
- ✓ Zoom



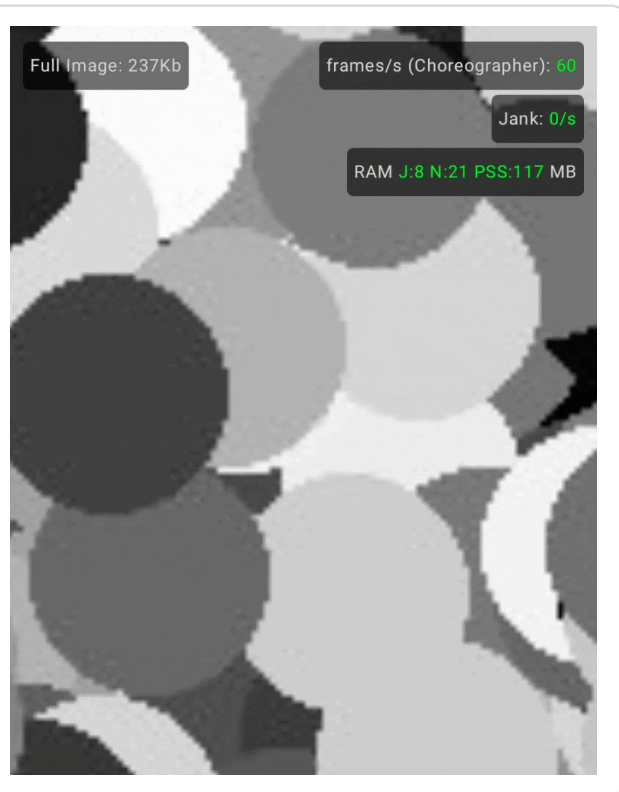
pan/zoom в лоб

test_image_small.jpg

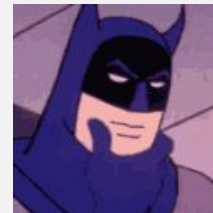
→ 1 000 x 1 000 px

→ 237 Kb

- ✓ Pan
- ✓ Zoom



- (?) Pan
- (?) Zoom



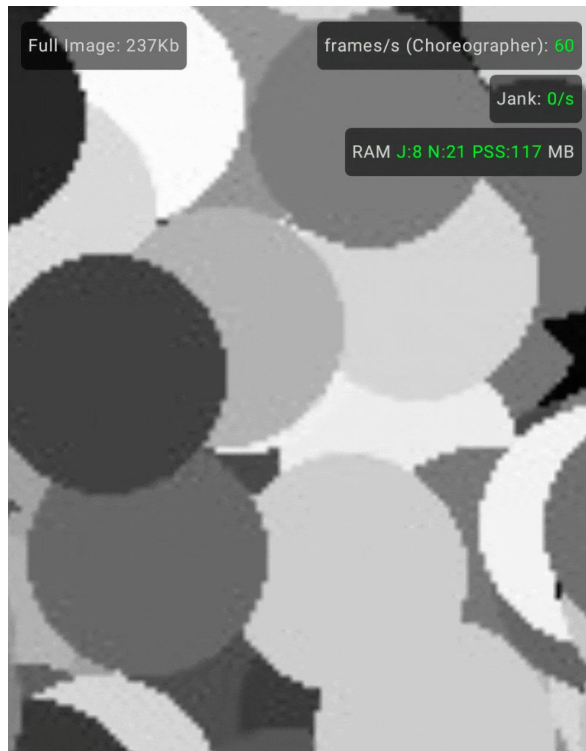
pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

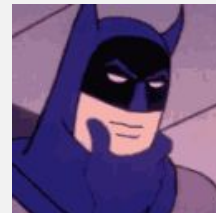
→ 237 Kb

- ✓ Pan
- ✓ Zoom



test_image_medium.jpg

- (?) Pan
- (?) Zoom



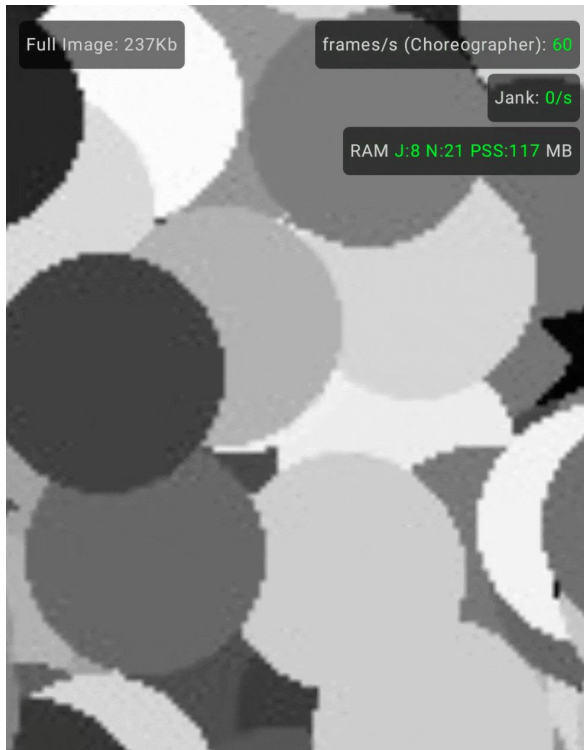
pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

→ 237 Kb

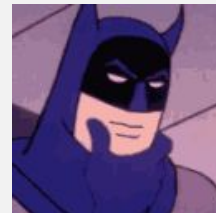
- ✓ Pan
- ✓ Zoom



test_image_medium.jpg

→ 8 000 x 8 000 px

- (?) Pan
- (?) Zoom



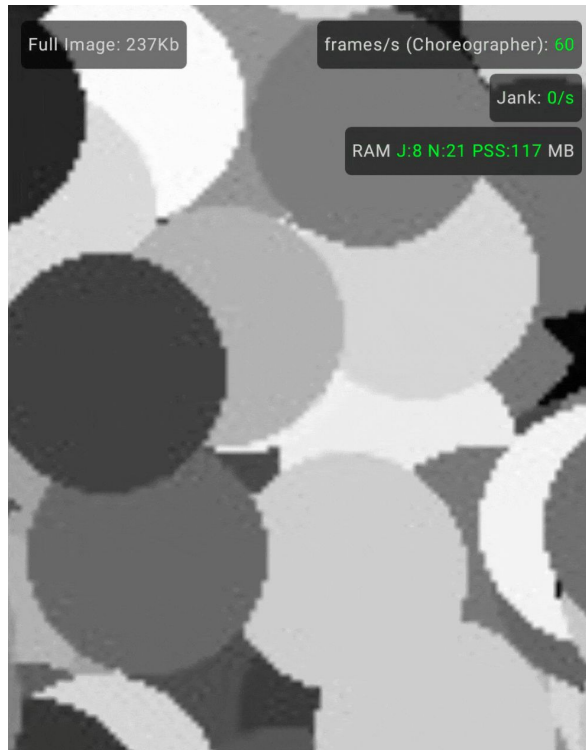
pan/zoom в лоб

test_image_small.jpg

→ 1 000 x 1 000 px

→ 237 Kb

- ✓ Pan
- ✓ Zoom

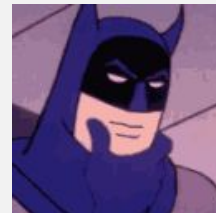


test_image_medium.jpg

→ 8 000 x 8 000 px

→ 13.2 Mb

- (?) Pan
- (?) Zoom

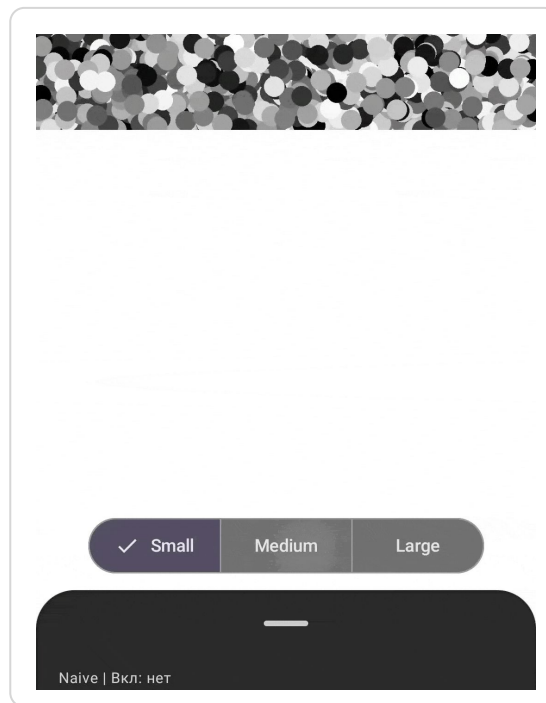


Загрузим большое изображение?

РЕАЛЬНЫЙ КРАШ



НО МЫ ЕГО ПЕРЕХВАТИМ



Ошибка краша: [?]

«в лоб» → краш

*без перехвата
исключения

Ошибка краша: [?]

«в лоб» → краш

*без перехвата
исключения



```
Image test_image_medium.jpg is open.  
Size: 13163661 bytes
```

Ошибка краша: [?]

«в лоб» → краш

*без перехвата
исключения



```
Image test_image_medium.jpg is open.  
Size: 13163661 bytes
```

ПАМЯТЬ БЫЛА ВЫДЕЛЕНА

Ошибка краша: Too large bitmap

«В лоб» → Canvas: too large bitmap (256 MB) → краш

*без перехвата
исключения



```
java.lang.RuntimeException: Canvas: trying to draw  
too large(256000000bytes) bitmap.  
    at RecordingCanvas.throwIfCannotDraw  
    at BitmapPainter.onDraw
```

Ошибка краша: Too large bitmap

«в лоб» → **Canvas: too large bitmap (256 MB)** → краш

*без перехвата
исключения



```
java.lang.RuntimeException: Canvas: trying to draw  
too large(256000000bytes) bitmap.  
    at RecordingCanvas.throwIfCannotDraw  
    at BitmapPainter.onDraw
```

```
ФАЗА 1 - COMPOSITION - 
| Image(bitmap = imageBitmap, ...) 
|
| [node tree зафиксирован, ждём Draw-фазу]
|
ФАЗА 2 - DRAW (Main Thread, triggered by Choreographer) 
| Painter.draw(canvas, size, ...)
|
| BitmapPainter.onDraw(canvas: Canvas, size: Size)
|
| android.graphics.Canvas.drawBitmap(bitmap, ...)
|
| RecordingCanvas.throwIfCannotDraw(bitmap: Bitmap) 
| Проверка:
|   bitmap.getAllocationByteCount() → 256 000 000 bytes
|   CANVAS_MAX_BITMAP_SIZE → 100 * 1024 * 1024 = 104 857 600 bytes
|   256 000 000 > 104 857 600 → TRUE
|
|
| RuntimeException("Canvas: trying to draw too large(256000000bytes) bitmap") 
```

А если взять изображение еще больше?

А если взять изображение еще больше?

30k×30k

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш

test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐛

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш



test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }
| требуется ~3.35 GiB под пиксели в native heap

▼ native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼ BitmapFactory.decodeStream() → null

?: error("Failed to decode test_image_large.jpg") ✖

| [наш null-check]

▼ IllegalStateException("Failed to decode test_image_large.jpg")

▼ перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш



test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }
| требуется ~3.35 GiB под пиксели в native heap

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖

| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш



test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }
| требуется ~3.35 GiB под пиксели в native heap

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш

test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐛

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → → → → → → → краш

test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✨
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

test_image_large.jpg (30k×30k) · Android 7- · теоретический сценарий

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) } 🐞
```

| Android 7-: пиксели хранятся в Java heap
| запрос ~3.4 ГБ через Java-аллокатор

▼
BitmapFactory.decodeStream()
| JVM пытается выделить byte[] размером 3 600 000 000
| → Java heap exhausted

▼
OutOfMemoryError: Failed to allocate a 3600000000-byte allocation ✨
| [брошена прямо из BitmapFactory, до нашего null-check дело не доходит]
| [AndroidRuntime]: FATAL EXCEPTION: main

▼
перехват ошибки → сообщение пользователю → Naive Small

[Draw-фаза не наступила - Composition упала раньше]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → **IllegalStateException/OutOfMemoryError** → краш

test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

test_image_large.jpg (30k×30k) · Android 7- · теоретический сценарий

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) } 🐞
```

| Android 7-: пиксели хранятся в Java heap
| запрос ~3.4 ГБ через Java-аллокатор

▼
BitmapFactory.decodeStream()

| JVM пытается выделить byte[] размером 3 600 000 000
| → Java heap exhausted

▼
OutOfMemoryError: Failed to allocate a 3600000000-byte allocation ✖
| [брошена прямо из BitmapFactory, до нашего null-check дело не доходит]
| [AndroidRuntime]: FATAL EXCEPTION: main

▼
перехват ошибки → сообщение пользователю → Naive Small

[Draw-фаза не наступила - Composition упала раньше]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → **IllegalStateException/OutOfMemoryError** → краш

test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

test_image_large.jpg (30k×30k) · Android 7- · теоретический сценарий

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) } 🐞
```

| Android 7-: пиксели хранятся в Java heap
| запрос ~3.4 ГБ через Java-аллокатор

▼
BitmapFactory.decodeStream()
| JVM пытается выделить byte[] размером 3 600 000 000
| → Java heap exhausted

▼
OutOfMemoryError: Failed to allocate a 3600000000-byte allocation ✖
| [брошена прямо из BitmapFactory, до нашего null-check дело не доходит]
| [AndroidRuntime]: FATAL EXCEPTION: main

▼
перехват ошибки → сообщение пользователю → Naive Small

[Draw-фаза не наступила - Composition упала раньше]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → **IllegalStateException/OutOfMemoryError** → краш



test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]



test_image_large.jpg (30k×30k) · Android 7- · теоретический сценарий

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) } 🐞
```

| Android 7-: пиксели хранятся в Java heap
| запрос ~3.4 ГБ через Java-аллокатор

▼
BitmapFactory.decodeStream()

| JVM пытается выделить byte[] размером 3 600 000 000
| → Java heap exhausted

▼
OutOfMemoryError: Failed to allocate a 3600000000-byte allocation ✖
| [брошена прямо из BitmapFactory, до нашего null-check дело не доходит]
| [AndroidRuntime]: FATAL EXCEPTION: main

▼
перехват ошибки → сообщение пользователю → Naive Small

[Draw-фаза не наступила - Composition упала раньше]

А если взять изображение еще больше?

30k×30k: ~3.4 ГБ → **IllegalStateException/OutOfMemoryError** → краш

test_image_large.jpg (30k×30k) · Android 8+

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) }  
| требуется ~3.35 GiB под пиксели в native heap
```

▼
native-аллокатор

| памяти нет - возвращает null, не бросает исключение

▼
BitmapFactory.decodeStream() → null

▼
?: error("Failed to decode test_image_large.jpg") ✖
| [наш null-check]

▼
IllegalStateException("Failed to decode test_image_large.jpg")

▼
перехват ошибки → сообщение пользователю → Naive Small

[Large не дошёл до Draw - декодер упал еще на выборе ассета]

test_image_large.jpg (30k×30k) · Android 7- · теоретический сценарий

ДЕКОДИРОВАНИЕ (IO dispatcher, background thread) 🐞

```
withContext(dispatchers.io) { BitmapFactory.decodeStream(it) } 🐞
```

| Android 7-: пиксели хранятся в Java heap
| запрос ~3.4 ГБ через Java-аллокатор

▼
BitmapFactory.decodeStream()
| JVM пытается выделить byte[] размером 3 600 000 000
| → Java heap exhausted

▼
OutOfMemoryError: Failed to allocate a 3600000000-byte allocation ✖
| [брошена прямо из BitmapFactory, до нашего null-check дело не доходит]
| [AndroidRuntime]: FATAL EXCEPTION: main

▼
перехват ошибки → сообщение пользователю → Naive Small

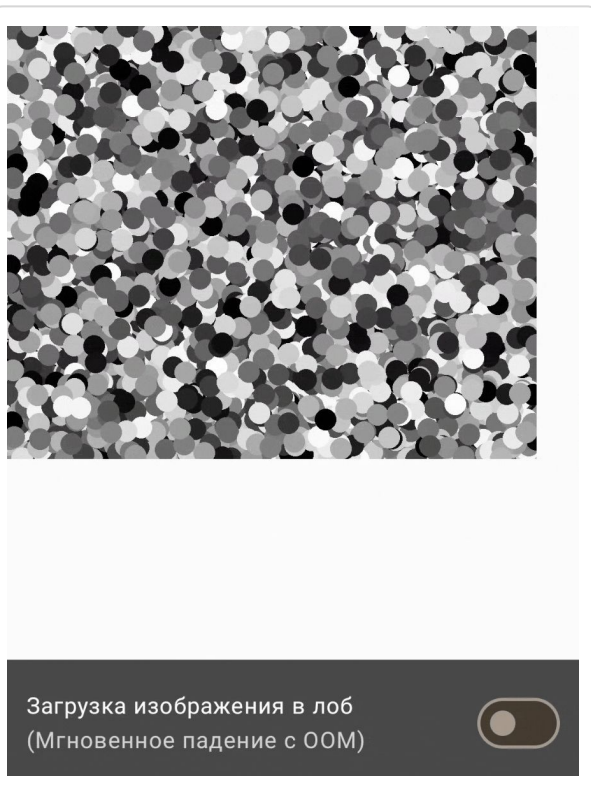
[Draw-фаза не наступила - Composition упала раньше]

В лоб — не наш вариант



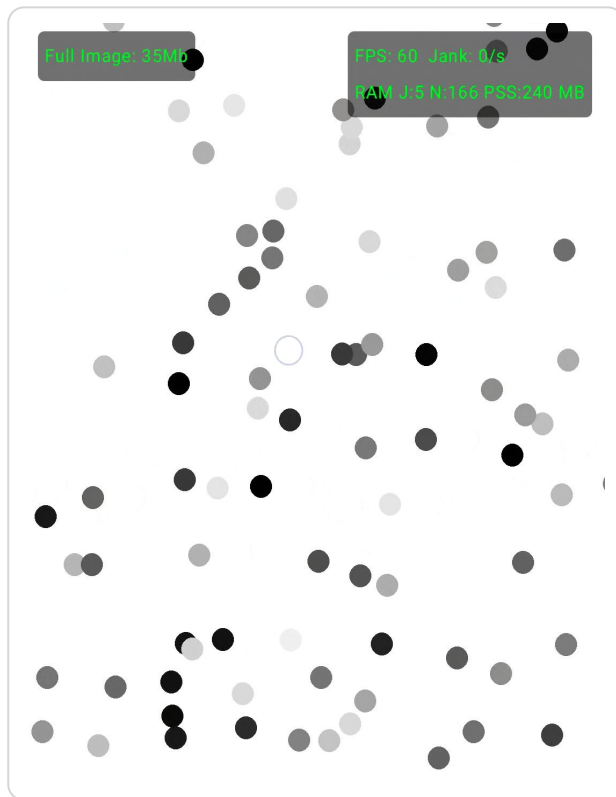
Сейчас — краш

SURF . RU



В конце — полетаем

SURF . RU





Знание и понимание

SURF . RU



План

SURF . RU

План

SURF . RU

[01] —————→ [02] —————→ [03]

План

SURF . RU

[01] —————→ [02] —————→ [03]

Куда уходит кадр?

План

SURF . RU

[01] —————→

Куда уходит кадр?

[02] —————→

**Враг №1:
Память**

[03]

План

SURF . RU

[01] —————→

Куда уходит кадр?

[02] —————→

**Враг №1:
Память**

[03]

**Враг №2:
Стоимость кадра**

Куда уходит кадр

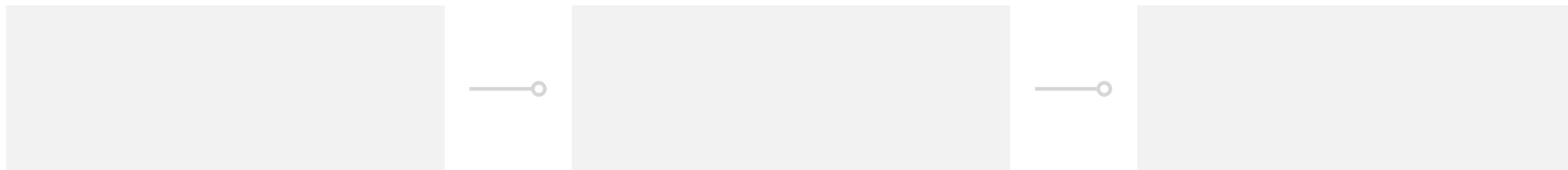
Как Compose рисует

SURF . RU

Разные оптимизации бьют
по разным фазам.

Как Compose рисует

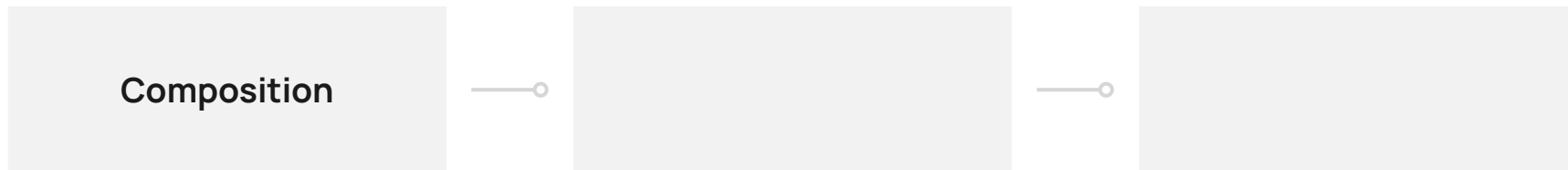
SURF . RU



**Разные оптимизации бьют
по разным фазам.**

Как Compose рисует

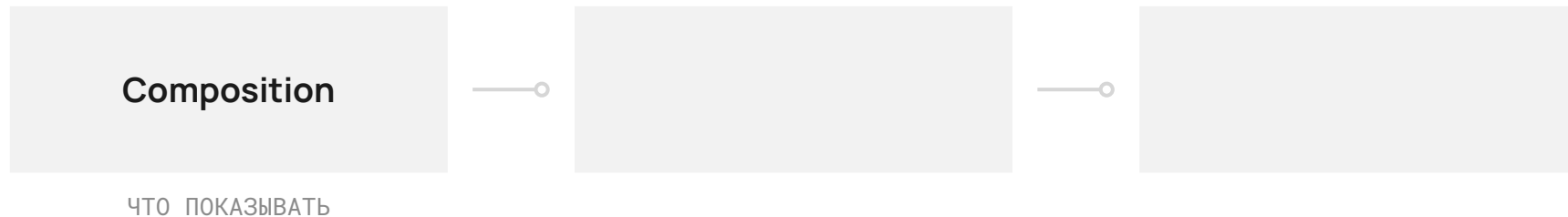
SURF . RU



**Разные оптимизации бьют
по разным фазам.**

Как Compose рисует

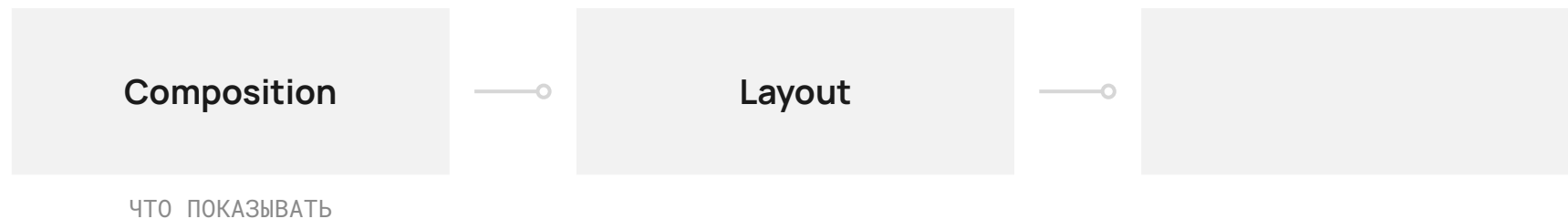
SURF . RU



Разные оптимизации бьют по разным фазам.

Как Compose рисует

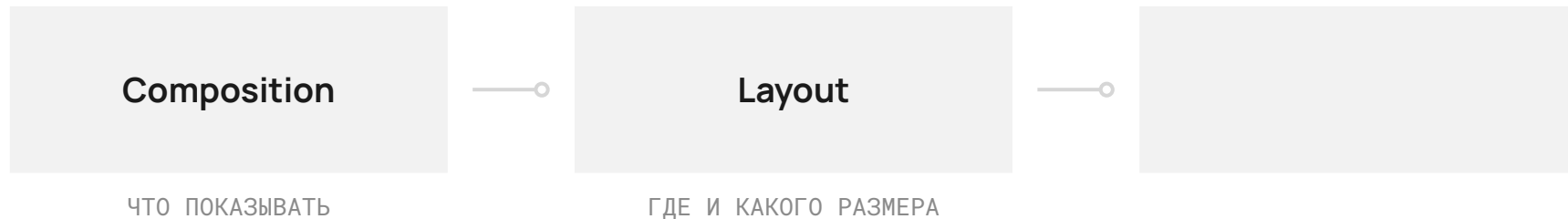
SURF . RU



Разные оптимизации бьют по разным фазам.

Как Compose рисует

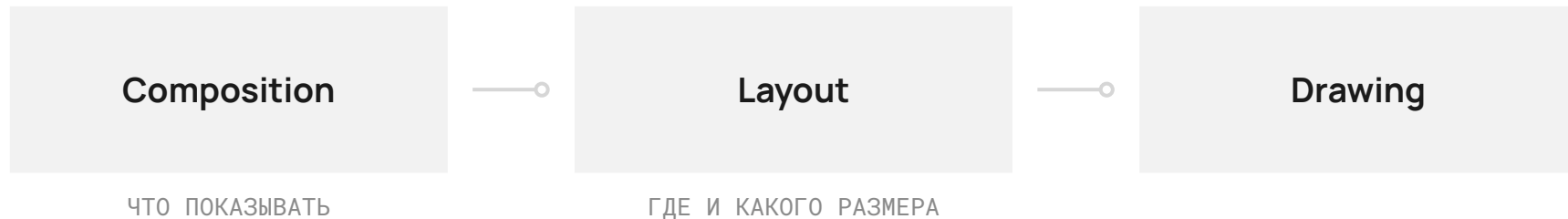
SURF . RU



Разные оптимизации бьют по разным фазам.

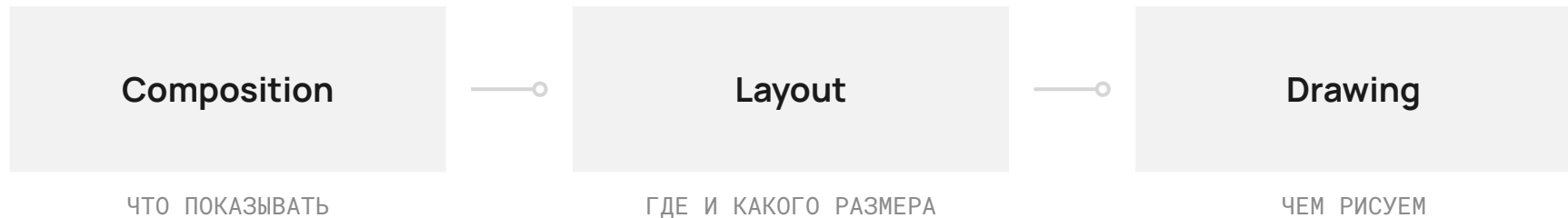
Как Compose рисует

SURF . RU



Разные оптимизации бьют по разным фазам.

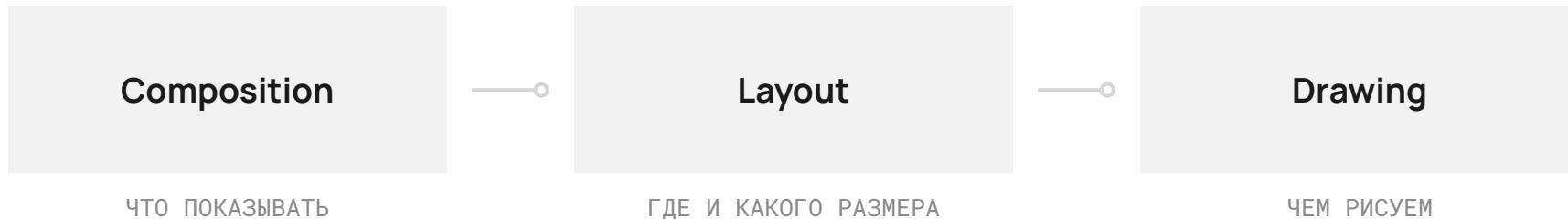
Как Compose рисует



Разные оптимизации бьют по разным фазам.

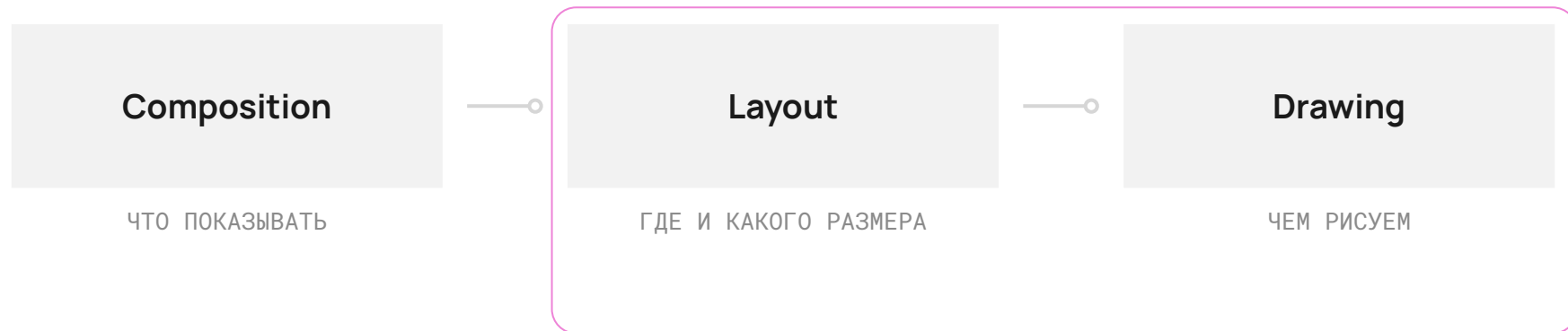
Фазы стоят по-разному

SURF . RU



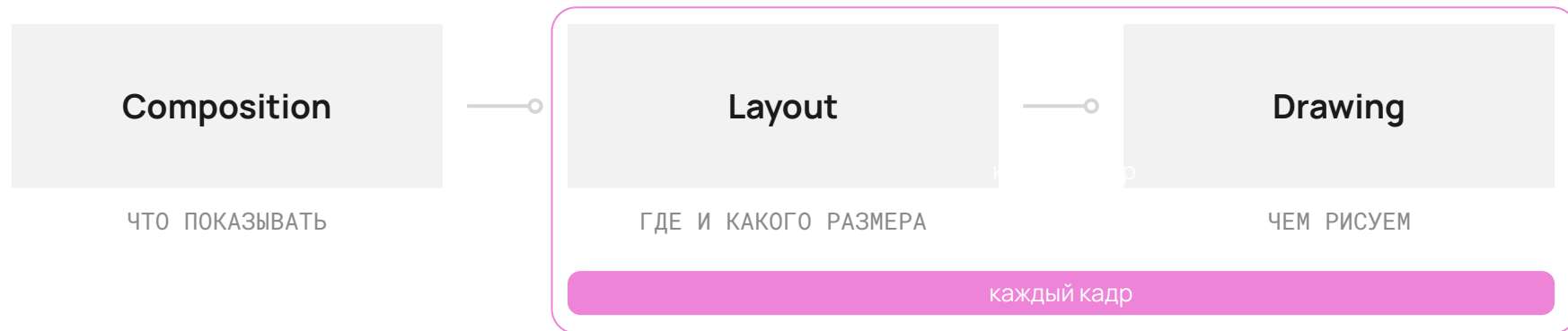
Фазы стоят по-разному

SURF . RU



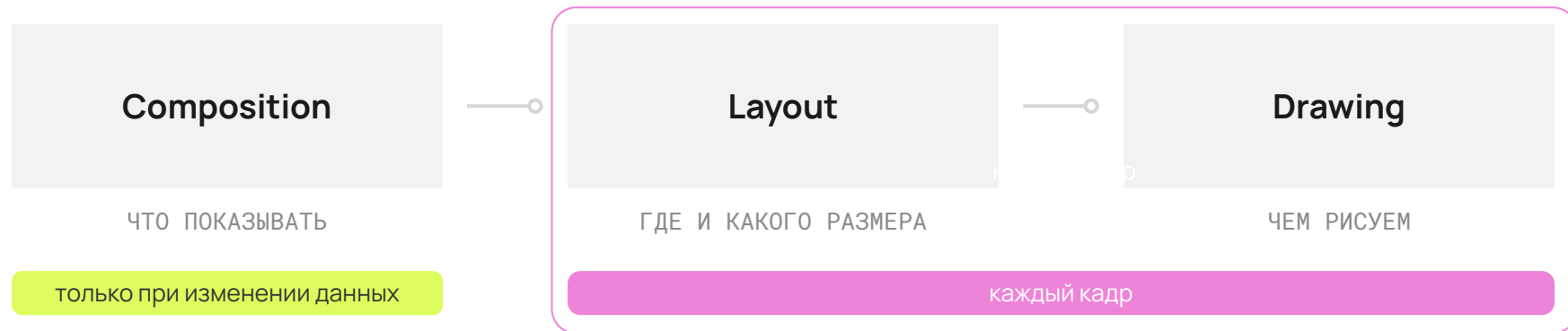
Фазы стоят по-разному

SURF . RU

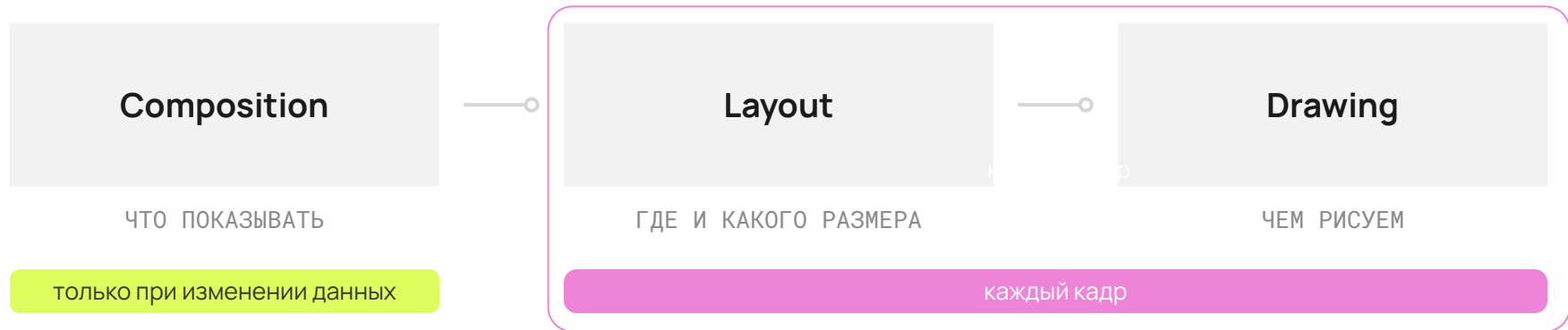


Фазы стоят по-разному

SURF . RU



Фазы стоят по-разному



Не дёргай Composition там,
где хватит одной отрисовки.

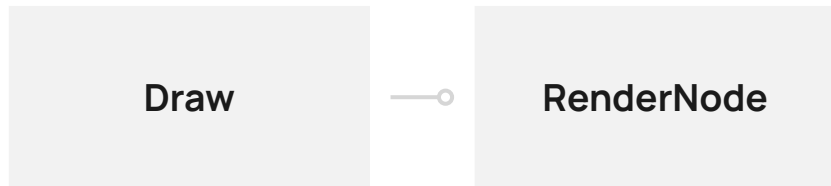
От draw до пикселя на экране

От draw до пикселя на экране

SURF . RU

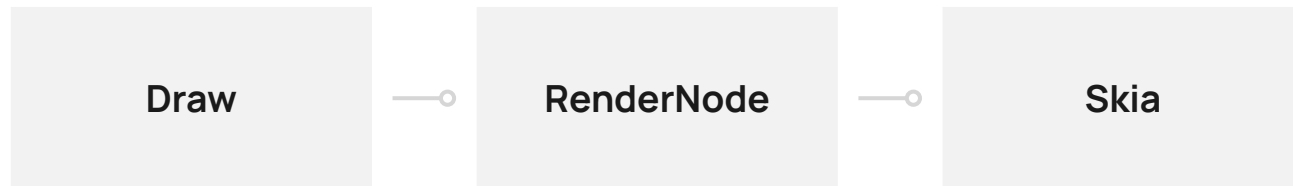
Draw

От draw до пикселя на экране



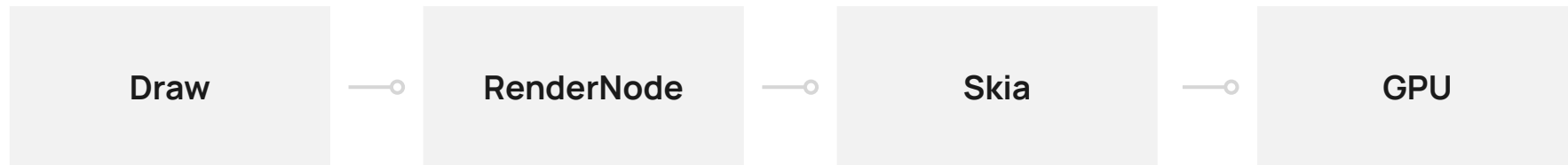
От draw до пикселя на экране

SURF . RU



От draw до пикселя на экране

SURF . RU



Бюджет кадра

Бюджет кадра

SURF . RU

16.6 мс

БЮДЖЕТ КАДРА

Бюджет кадра

SURF . RU

16.6 мс

БЮДЖЕТ КАДРА

60 Гц

16.6 мс

Бюджет кадра

SURF . RU

16.6 мс

БЮДЖЕТ КАДРА

1000 мс ÷ Гц:

60 Гц

16.6 мс

90 Гц

11.1 мс

120 Гц

8.3 мс

144 Гц

6.9 мс

Бюджет кадра

16.6 мс

БЮДЖЕТ КАДРА

**Не уложился = кадр пропущен
(глаз видит рывок)**

1000 мс ÷ Гц:

60 Гц

16.6 мс

90 Гц

11.1 мс

120 Гц

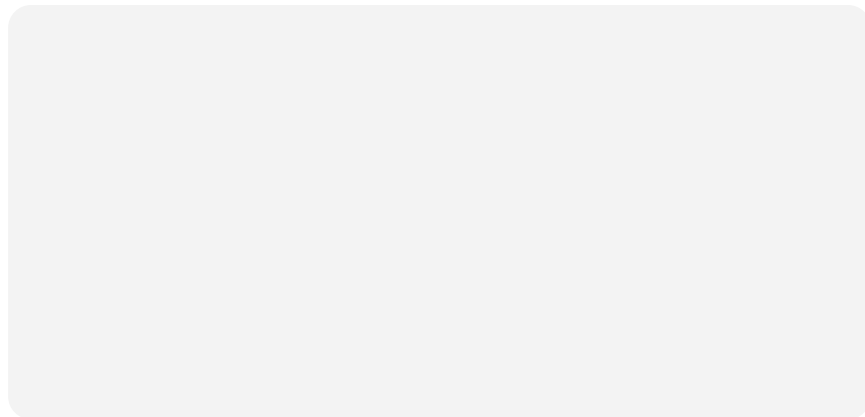
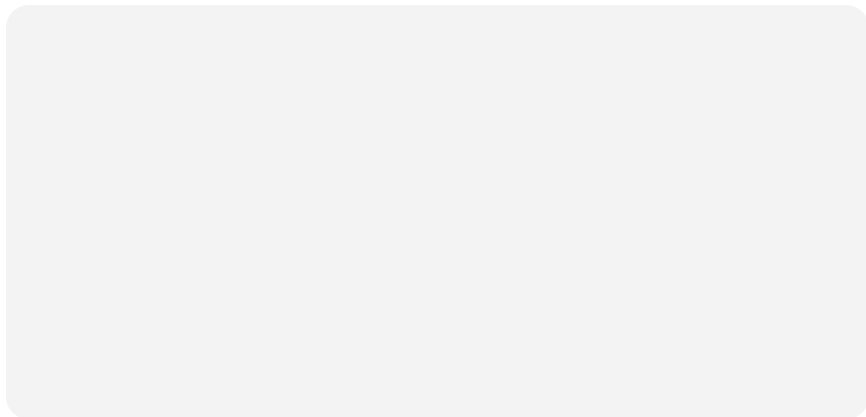
8.3 мс

144 Гц

6.9 мс

Кто твой враг?

SURF . RU



Разные звери → разные инструменты → разные фазы

Кто твой враг?

SURF . RU

Враг №1

Враг №2

Разные звери → разные инструменты → разные фазы

Кто твой враг?

SURF . RU

Враг №1

Разные звери → разные инструменты → разные фазы

Кто твой враг?

SURF . RU

Враг №1

Память

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает
в RAM приложения.

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает
в RAM приложения.

50 000 × 50 000

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает
в RAM приложения.

$50\ 000 \times 50\ 000 \times 4 \text{ байта} \approx 10 \text{ ГБ}$

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает
в RAM приложения.

$50\ 000 \times 50\ 000 \times 4$ байта ≈ 10 ГБ

Враг №2

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает в RAM приложения.

$50\ 000 \times 50\ 000 \times 4 \text{ байта} \approx 10 \text{ ГБ}$

Враг №2

Стоимость кадра

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает в RAM приложения.

$50\ 000 \times 50\ 000 \times 4 \text{ байта} \approx 10 \text{ ГБ}$

Враг №2

Стоимость кадра

Работа не влезает в 16.6 мс

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает в RAM приложения.

50 000 × 50 000 × 4 байта ≈ 10 ГБ

Враг №2

Стоимость кадра

Работа не влезает в 16.6 мс — затык на CPU

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает в RAM приложения.

$50\ 000 \times 50\ 000 \times 4 \text{ байта} \approx 10 \text{ ГБ}$

Враг №2

Стоимость кадра

Работа не влезает в 16.6 мс — затык на CPU или на GPU.

Разные звери → разные инструменты → разные фазы

Кто твой враг?

Враг №1

Память

Декодированный битмап не влезает в RAM приложения.

$50\ 000 \times 50\ 000 \times 4$ байта ≈ 10 ГБ

Враг №2

Стоимость кадра

Работа не влезает в 16.6 мс — затык на CPU или на GPU.

рекомпозиция · overdraw

Разные звери → разные инструменты → разные фазы

К гадалке не ходи — на цифры смотри

[01]

Logcat

FPS · RAM · декодер · пул

[02]

Perfetto + Profile GPU Rendering

frame timeline

[03]

gfxinfo

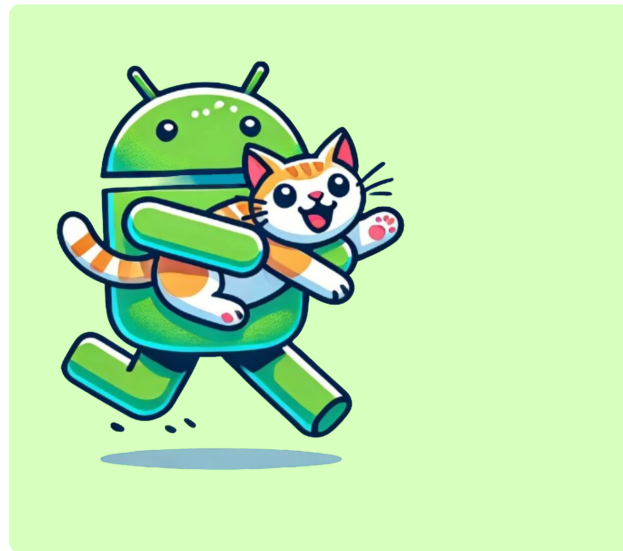
процентили · Slow bitmap
uploads

[04]

Layout Inspector

счётчики рекомпозиций

Logcat — основной источник цифр



Logcat — основной источник цифр



Кастомный
оверлей на экране



Logcat — основной источник цифр



Кастомный
оверлей на экране

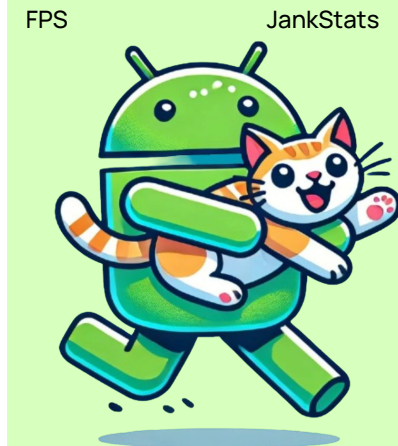
FPS



Logcat — ОСНОВНОЙ ИСТОЧНИК ЦИФР



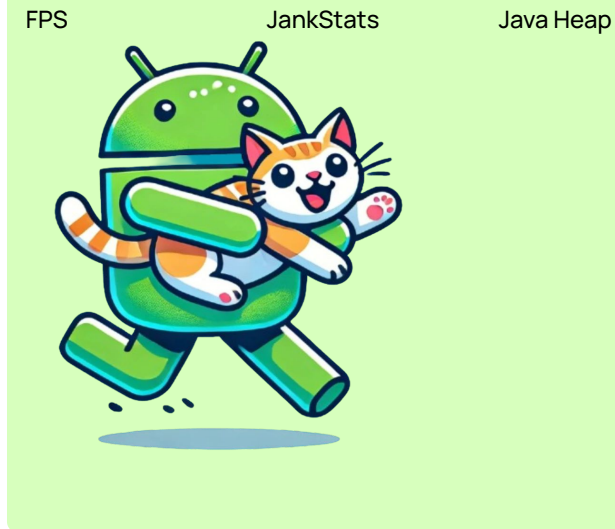
Кастомный
оверлей на экране



Logcat — ОСНОВНОЙ ИСТОЧНИК ЦИФР



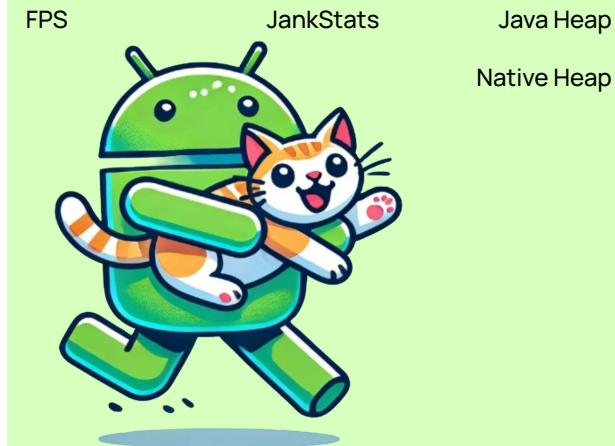
Кастомный
оверлей на экране



Logcat — ОСНОВНОЙ ИСТОЧНИК ЦИФР



Кастомный
оверлей на экране



Logcat — ОСНОВНОЙ ИСТОЧНИК ЦИФР



Кастомный
оверлей на экране



Logcat — основной источник цифр



Кастомный
оверлей на экране



Логи на каждый
тайл

FPS

JankStats

Java Heap

Native Heap

Proportional Set
Size



Logcat — основной источник цифр



Кастомный
оверлей на экране



Логи на каждый
тайл

FPS

JankStats

Java Heap

Native Heap

Proportional Set
Size



Decode avg

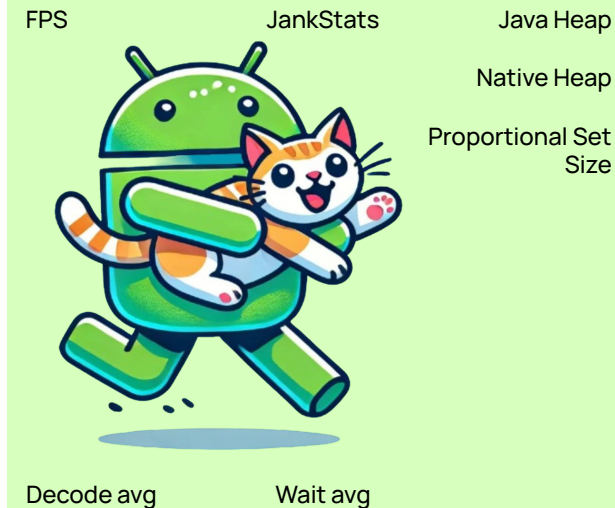
Logcat — основной источник цифр



Кастомный
оверлей на экране



Логи на каждый
тайл



Logcat — основной источник цифр



Кастомный
оверлей на экране



Логи на каждый
тайл

FPS

JankStats

Java Heap

Native Heap

Proportional Set
Size



Decode avg

Wait avg

Pool hit-rate

Logcat — основной источник цифр



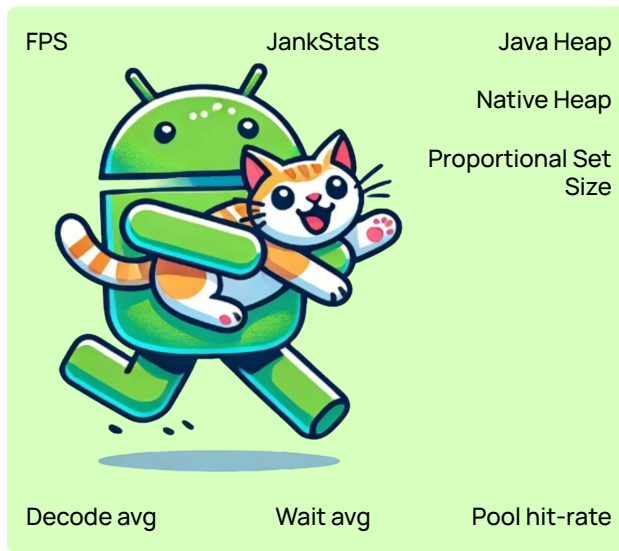
Кастомный
оверлей на экране



Логи на каждый
тайл



Основной источник
цифр по памяти
и декодеру



Perfetto — таймлайн системы

Perfetto



Perfetto — таймлайн системы



Системный
профайлер
от Google

Perfetto



Perfetto — таймлайн системы



Системный
профайлер
от Google

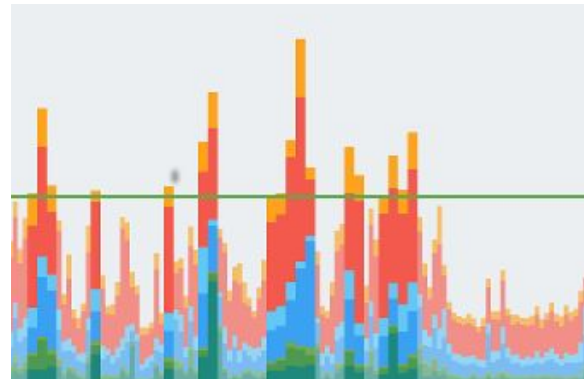


Пишет таймлайн:
фреймы · потоки

Perfetto



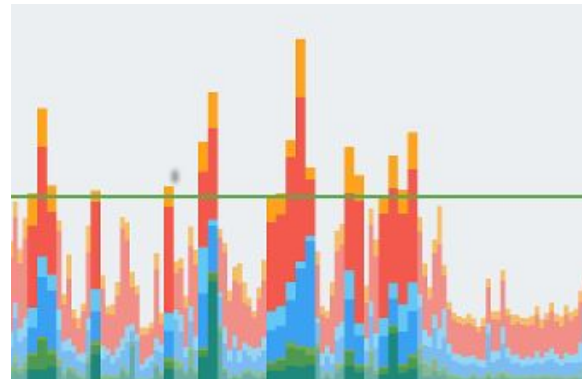
gfxinfo — статистика кадров от Android



gfxinfo — статистика кадров от Android



Процентили
времени рендера



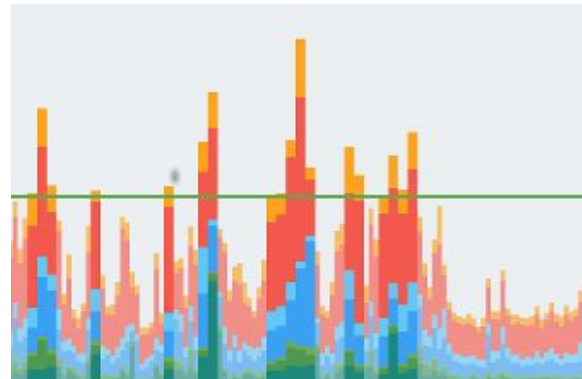
gfxinfo — статистика кадров от Android



Процентили
времени рендера



Janky frames%



gfxinfo — статистика кадров от Android



Процентили
времени рендера

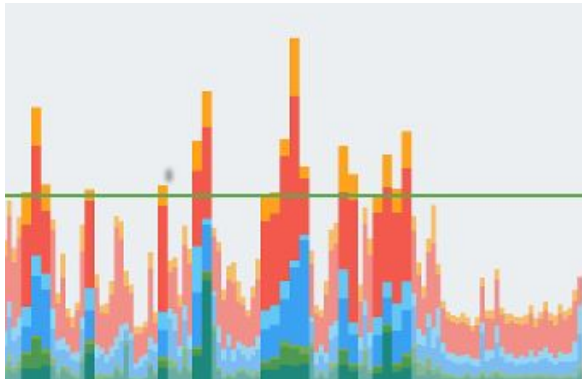


Janky frames%

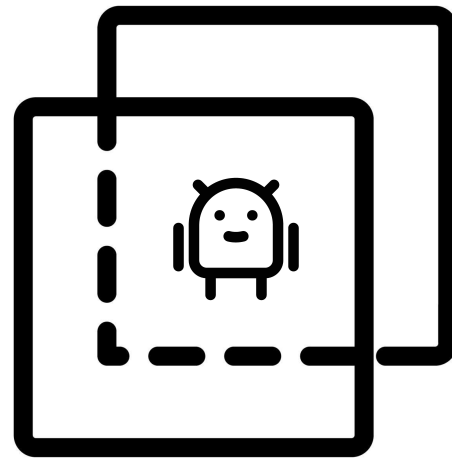


Slow bitmap uploads

давление текстур на GPU



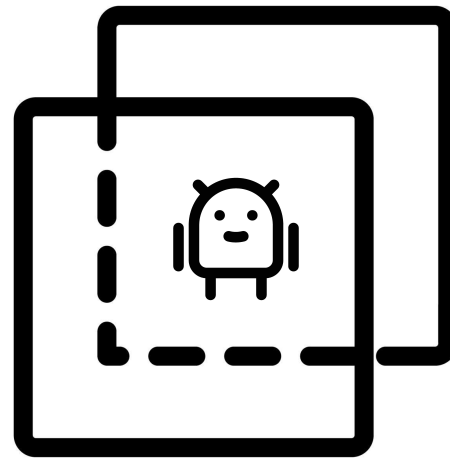
Layout Inspector — счётчик рекомпозиций



Layout Inspector — счётчик рекомпозиций



Встроен в Android
Studio



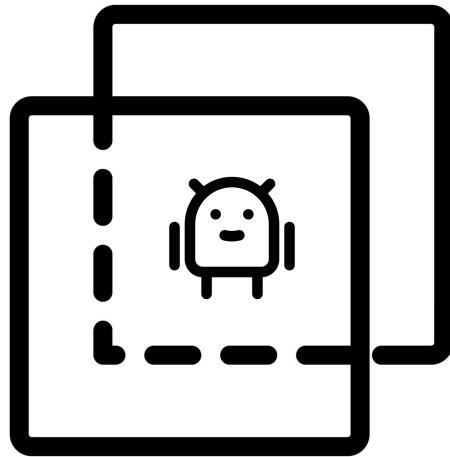
Layout Inspector — счётчик рекомпозиций



Встроен в Android
Studio



Счётчик
рекомпозиций
каждого composable



Измерим эффективность?

ОБЪЕКТИВНОСТЬ ДАННЫХ

Изолируемость
от других приложений

Повторяемость сценария

Изолируемость прогонов

Изолируемость от процессов

ОБЪЕКТИВНОСТЬ ДАННЫХ

Изолируемость
от других приложений

Повторяемость сценария

Изолируемость прогонов

Повторение – мать учения

ОБЪЕКТИВНОСТЬ ДАННЫХ

Изолируемость
от других приложений

Повторяемость сценария

Изолируемость прогонов

Изолируемость прогонов

ОБЪЕКТИВНОСТЬ ДАННЫХ

Изолируемость
от других приложений

Повторяемость сценария

Изолируемость прогонов

Сценарии

Эталон
PAN

Эталон
ZOOM

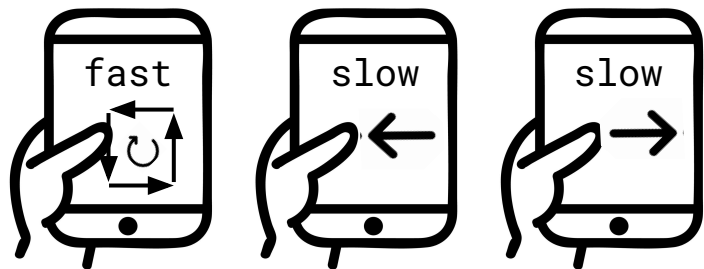
ОРУЖИЕ
1

ОРУЖИЕ
2

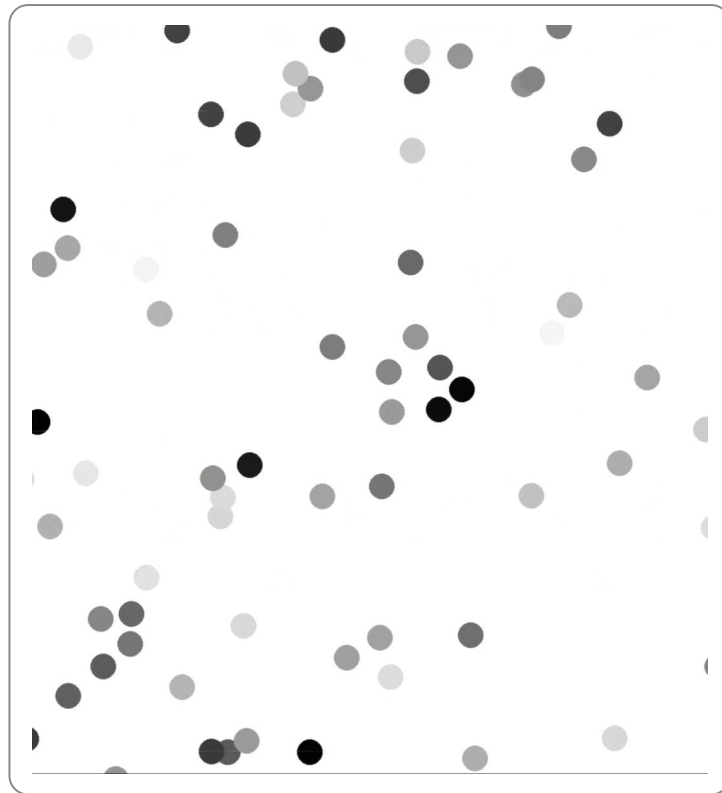
ОРУЖИЕ
3

Сценарий А

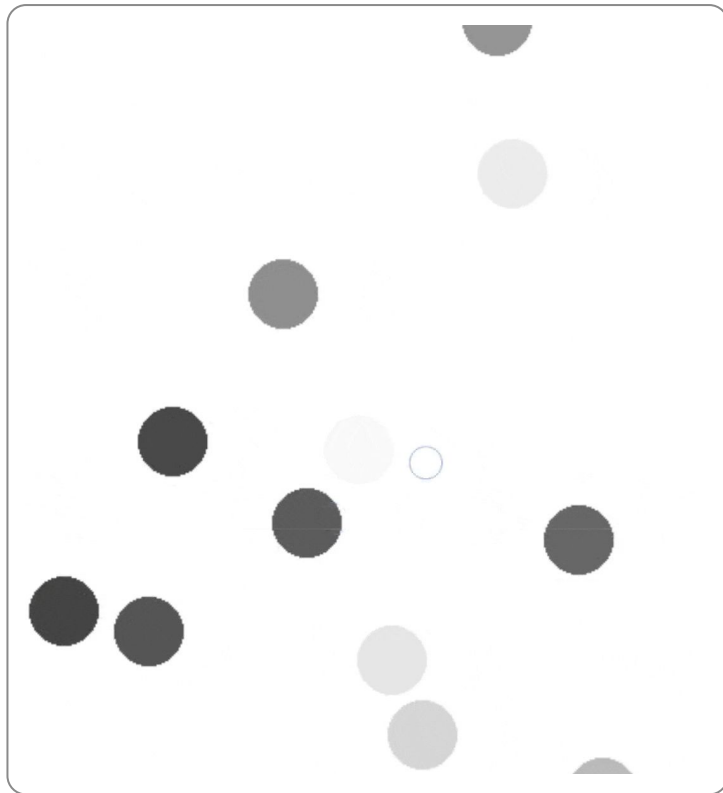
Эталон
PAN



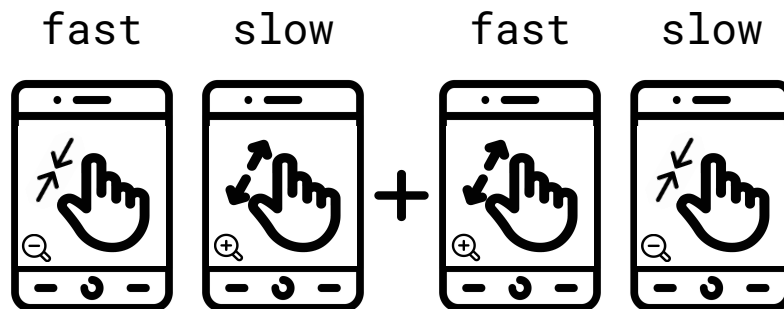
Источник изображения: Теплов



Сценарий В



Эталон
ZOOM



Источник изображения: Tenor

К гадалке не ходи — на цифры смотри

[01]

Logcat

FPS · RAM · декодер · пул

[02]

Perfetto + Profile GPU Rendering

frame timeline

[03]

gfxinfo

процентили · Slow bitmap
uploads

[04]

Layout Inspector

счётчики рекомпозиций

2 ЭТАЛОННЫХ СЦЕНАРИЯ

оптимизированный pan + оптимизированный zoom

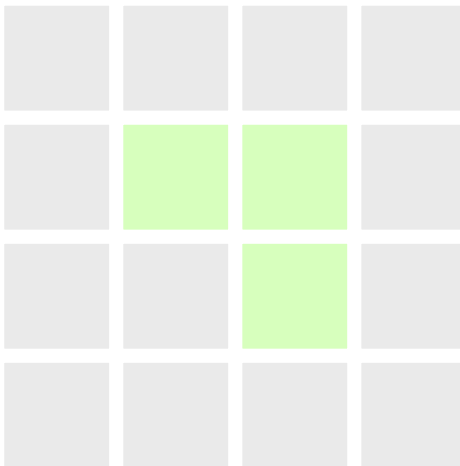
3 ПРОВЕРОЧНЫХ СЦЕНАРИЯ

для каждого оружия

Каждый фикс — измерен, а не придуман.

Враг №1: память

Декодируем только то, что видим



ВИДИМЫЕ ТАЙЛЫ

Декодируем только то, что видим

BitmapRegionDecoder

достаёт нужный кусок, не открывая всё
изображение.

Viewport culling

всё, что за экраном, не декодируем
и не рисуем



ВИДИМЫЕ ТАЙЛЫ

Как это выглядит в коде

```
private val freeDecoders = Channel<BitmapRegionDecoder>(
    capacity = 3,
    onUndeliveredElement = ::returnDecoder,
)

init {
    repeat(3) { returnDecoder(createDecoder()) }
}

suspend fun decodeTile(rect: Rect, sampleSize: Int): Bitmap? {
    val decoder = freeDecoders.receive() // Если все заняты, ждём.
    try {
        val opts = BitmapFactory.Options().apply {
            inSampleSize = sampleSize
        }
        return decoder.decodeRegion(rect, opts)
    } finally {
        returnDecoder(decoder)
    }
}

private fun returnDecoder(decoder: BitmapRegionDecoder) {
    if (!freeDecoders.trySend(decoder).isSuccess) {
        decoder.recycle()
    }
}
```

Как это выглядит в коде

```
private val freeDecoders = Channel<BitmapRegionDecoder>(
    capacity = 3,
    onUndeliveredElement = ::returnDecoder,
)

init {
    repeat(3) { returnDecoder(createDecoder()) }
}

suspend fun decodeTile(rect: Rect, sampleSize: Int): Bitmap? {
    val decoder = freeDecoders.receive() // Если все заняты, ждём.
    try {
        val opts = BitmapFactory.Options().apply {
            inSampleSize = sampleSize
        }
        return decoder.decodeRegion(rect, opts)
    } finally {
        returnDecoder(decoder)
    }
}

private fun returnDecoder(decoder: BitmapRegionDecoder) {
    if (!freeDecoders.trySend(decoder).isSuccess) {
        decoder.recycle()
    }
}
```

ТРИ НЕЗАВИСИМЫХ ДЕКОДЕРА

Как это выглядит в коде

```
private val freeDecoders = Channel<BitmapRegionDecoder>(
    capacity = 3,
    onUndeliveredElement = ::returnDecoder,
)

init {
    repeat(3) { returnDecoder(createDecoder()) }
}

suspend fun decodeTile(rect: Rect, sampleSize: Int): Bitmap? {
    val decoder = freeDecoders.receive() // Если все заняты, ждём.
    try {
        val opts = BitmapFactory.Options().apply {
            inSampleSize = sampleSize
        }
        return decoder.decodeRegion(rect, opts)
    } finally {
        returnDecoder(decoder)
    }
}

private fun returnDecoder(decoder: BitmapRegionDecoder) {
    if (!freeDecoders.trySend(decoder).isSuccess) {
        decoder.recycle()
    }
}
```

ТРИ НЕЗАВИСИМЫХ ДЕКОДЕРА

CHANNEL КАК СЕМАФОР

Как это выглядит в коде



```
private val freeDecoders = Channel<BitmapRegionDecoder>(
    capacity = 3,
    onUndeliveredElement = ::returnDecoder,
)

init {
    repeat(3) { returnDecoder(createDecoder()) }
}

suspend fun decodeTile(rect: Rect, sampleSize: Int): Bitmap? {
    val decoder = freeDecoders.receive() // Если все заняты, ждём.
    try {
        val opts = BitmapFactory.Options().apply {
            inSampleSize = sampleSize
        }
        return decoder.decodeRegion(rect, opts)
    } finally {
        returnDecoder(decoder)
    }
}

private fun returnDecoder(decoder: BitmapRegionDecoder) {
    if (!freeDecoders.trySend(decoder).isSuccess) {
        decoder.recycle()
    }
}
```

ТРИ НЕЗАВИСИМЫХ ДЕКОДЕРА



```
val viewport = ImageRect(
    left = (-offset.x / scale).coerceAtLeast(0f),
    top = (-offset.y / scale).coerceAtLeast(0f),
    right = (screenWidth - offset.x) / scale,
    bottom = (screenHeight - offset.y) / scale,
)

val neededTiles = grid.visibleTiles(viewport, sampleSize)
```

CHANNEL КАК СЕМАФОР

Как это выглядит в коде

```
private val freeDecoders = Channel<BitmapRegionDecoder>(
    capacity = 3,
    onUndeliveredElement = ::returnDecoder,
)

init {
    repeat(3) { returnDecoder(createDecoder()) }
}

suspend fun decodeTile(rect: Rect, sampleSize: Int): Bitmap? {
    val decoder = freeDecoders.receive() // Если все заняты, ждём.
    try {
        val opts = BitmapFactory.Options().apply {
            inSampleSize = sampleSize
        }
        return decoder.decodeRegion(rect, opts)
    } finally {
        returnDecoder(decoder)
    }
}

private fun returnDecoder(decoder: BitmapRegionDecoder) {
    if (!freeDecoders.trySend(decoder).isSuccess) {
        decoder.recycle()
    }
}
```

ТРИ НЕЗАВИСИМЫХ ДЕКОДЕРА

```
val viewport = ImageRect(
    left = (-offset.x / scale).coerceAtLeast(0f),
    top = (-offset.y / scale).coerceAtLeast(0f),
    right = (screenWidth - offset.x) / scale,
    bottom = (screenHeight - offset.y) / scale,
)

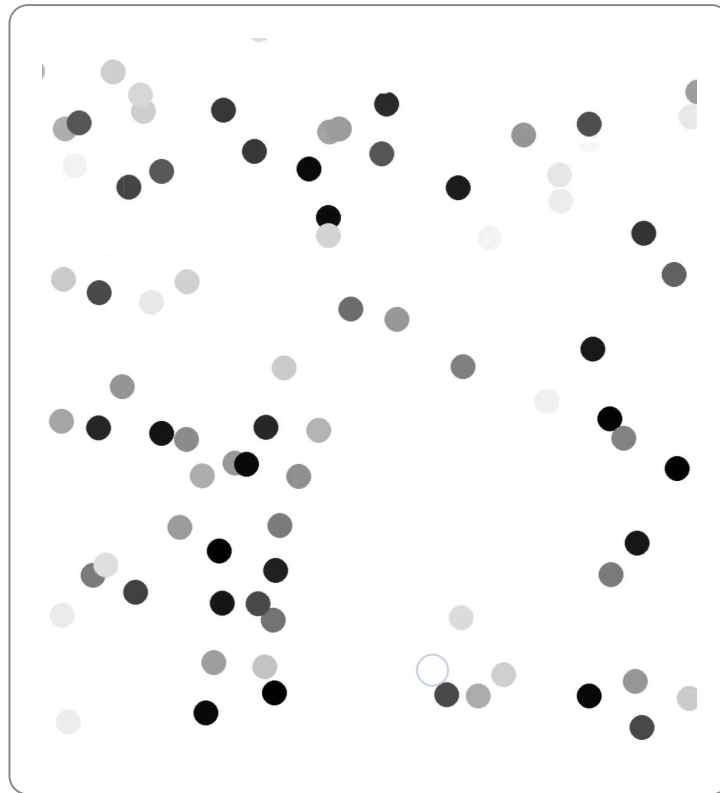
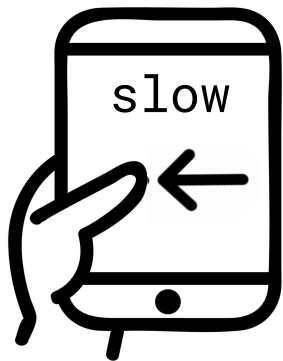
val neededTiles = grid.visibleTiles(viewport, sampleSize)
```

Передаем на загрузку только видимые тайлы

CHANNEL КАК СЕМАФОР

Сценарий: Culling=OFF

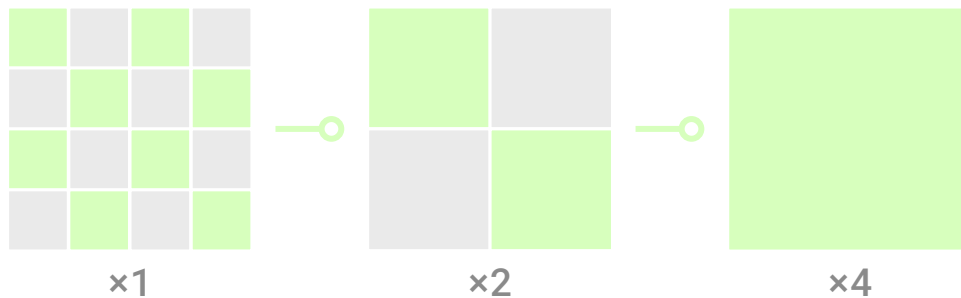
Медленные горизонтальные
свайпы от **левого** края
изображения **до** правого



Метрики

Метрика	Culling=ON (A)	Culling=OFF	Δ
RAM Native max	162 МБ	1 673 МБ	×10
PSS max	256 МБ	2 069 МБ	×8.1
Pool hit-rate	98.8 %	1.2 %	-97.6 п.п.
Wait avg	10.2 мс	7 811 мс	×781
Janky frames	1.16%	2.21%	×1.9

Меньше зум — меньше деталей

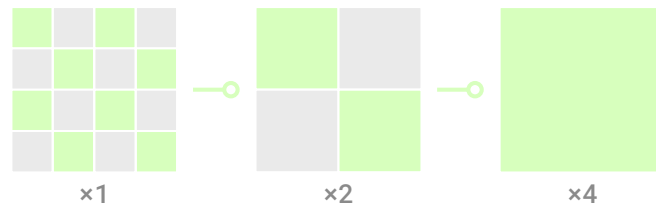


inSampleSize: дальше зум → грубее тайл

Меньше зум — меньше деталей

Субсэмплинг (inSampleSize)

декодируем сразу уменьшенным —
полный битмап в память не попадает.



inSampleSize: дальше зум → грубее тайл

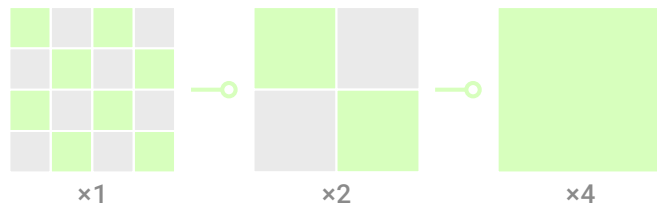
Меньше зум — меньше деталей

Субсэмплинг (inSampleSize)

декодируем сразу уменьшенным —
полный битмап в память не попадает.

LOD по уровню зума

дальше отъехал → больше inSampleSize → грубее тайл



inSampleSize: дальше зум → грубее тайл

Как это выглядит в коде



```
// LOD зависит от масштаба и уже применённого уровня.  
// Разные пороги входа и выхода защищают от дрожания.  
val desiredSampleSize = when (currentSampleSize) {  
    1 -> if (scale < 0.45f) 2 else 1  
    2 -> when {  
        scale > 0.55f -> 1  
        scale < 0.22f -> 4  
        else -> 2  
    }  
    4 -> when {  
        scale > 0.28f -> 2  
        scale < 0.11f -> 8  
        else -> 4  
    }  
    else -> if (scale > 0.14f) 4 else 8  
}  
  
// Новый уровень применяем после трёх одинаковых запросов.
```

Как это выглядит в коде



```
// LOD зависит от масштаба и уже применённого уровня.  
// Разные пороги входа и выхода защищают от дрожания.  
val desiredSampleSize = when (currentSampleSize) {  
    1 -> if (scale < 0.45f) 2 else 1  
    2 -> when {  
        scale > 0.55f -> 1  
        scale < 0.22f -> 4  
        else -> 2  
    }  
    4 -> when {  
        scale > 0.28f -> 2  
        scale < 0.11f -> 8  
        else -> 4  
    }  
    else -> if (scale > 0.14f) 4 else 8  
}  
  
// Новый уровень применяем после трёх одинаковых запросов.
```

LOD

просто when по зуму с гистерезисом

Как это выглядит в коде



```
// LOD зависит от масштаба и уже применённого уровня.  
// Разные пороги входа и выхода защищают от дрожания.  
val desiredSampleSize = when (currentSampleSize) {  
    1 -> if (scale < 0.45f) 2 else 1  
    2 -> when {  
        scale > 0.55f -> 1  
        scale < 0.22f -> 4  
        else -> 2  
    }  
    4 -> when {  
        scale > 0.28f -> 2  
        scale < 0.11f -> 8  
        else -> 4  
    }  
    else -> if (scale > 0.14f) 4 else 8  
}  
  
// Новый уровень применяем после трёх одинаковых запросов.
```

LOD

просто when по зуму с гистерезисом

inSampleSize — просьба, не приказ

декодер округляет до степени двойки вниз

Как это выглядит в коде



```
// LOD зависит от масштаба и уже применённого уровня.  
// Разные пороги входа и выхода защищают от дрожания.  
val desiredSampleSize = when (currentSampleSize) {  
    1 -> if (scale < 0.45f) 2 else 1  
    2 -> when {  
        scale > 0.55f -> 1  
        scale < 0.22f -> 4  
        else -> 2  
    }  
    4 -> when {  
        scale > 0.28f -> 2  
        scale < 0.11f -> 8  
        else -> 4  
    }  
    else -> if (scale > 0.14f) 4 else 8  
}  
  
// Новый уровень применяем после трёх одинаковых запросов.
```

LOD

просто when по зуму с гистерезисом

inSampleSize — просьба, не приказ

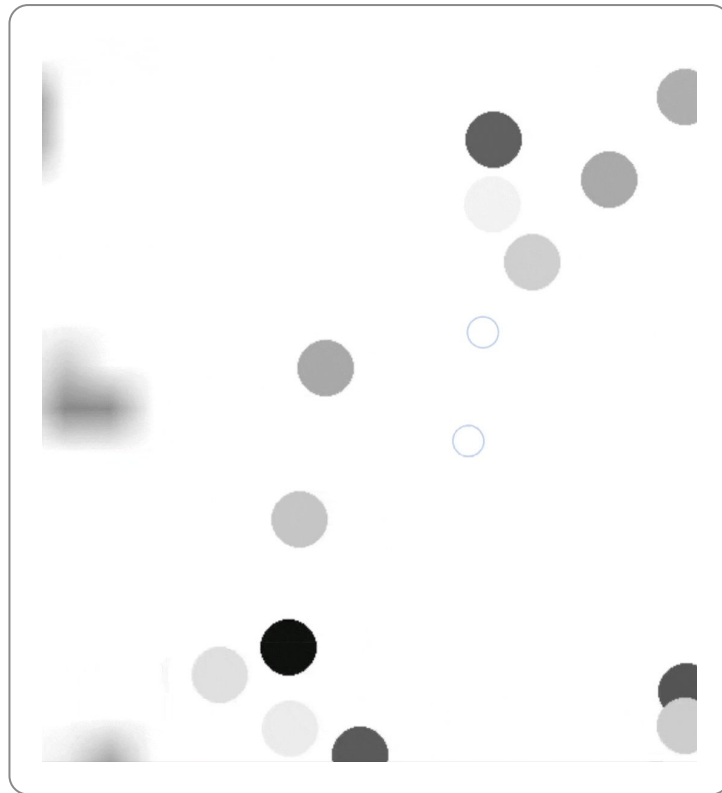
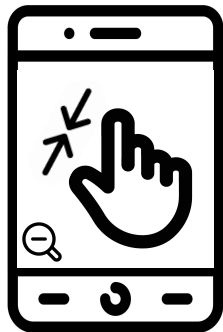
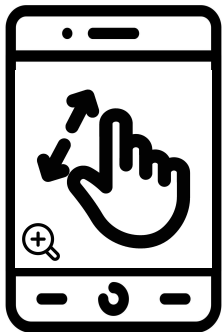
декодер округляет до степени двойки вниз

Передавать ровно 1, 2, 4, 8

иначе получим не то, что просили

Сценарий: LOD=OFF

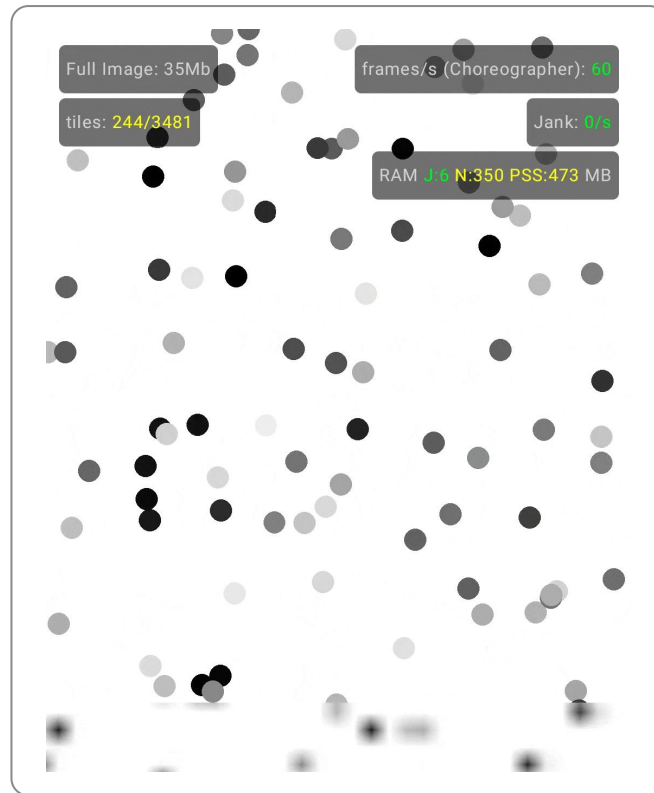
Приближаем изображение до $scale \approx 2$
Отдаляем изображение до $scale \approx 0.1$



Метрики

Метрика	LOD=ON (B)	LOD=OFF	Δ
RAM Native max	206 МБ	774 МБ	×3.8
Wait avg	242 мс	1 645 мс	×6.8
99 percentile	17 мс	28 мс	×1.6

Используем память с умом



Используем память с умом

Пул битмапов

освободившуюся плитку
переиспользуем, а не аллоцируем заново

ЗАГРУЗИТЬ ВСЮ КАРТИНКУ В ПАМЯТЬ	
СОЗДАВАТЬ НОВЫЙ БИТМАП ДЛЯ КАЖДОГО ТАЙЛА	
ПЕРЕИСПОЛЬЗОВАТЬ БИТМАПЫ ИЗ ПУЛА	
ПОДЕРЖАТЬ БИТМАП ТРИ КАДРА И ВЫДАТЬ ЗА НОВЫЙ	

Как это выглядит в коде: пул битмапов



```
val opts = BitmapFactory.Options().apply {
    inMutable = true
    inSampleSize = sampleSize
    inBitmap = pool.acquire(expectedW, expectedH)
}

val bitmap = decoder.decodeRegion(tileRect, opts)
```



```
private val buckets: LinkedHashMap<Long, ArrayDeque<Bitmap>> =
    LinkedHashMap(0, LOAD_FACTOR, true)

@Synchronized
fun acquire(width: Int, height: Int): Bitmap? {
    val bitmap = buckets[key(width, height)]?.removeFirstOrNull()
    ?: return null
    totalBytes -= bytesOf(width, height)
    return bitmap
}

@Synchronized
fun recycle(bitmap: Bitmap) {
    val bytes = bytesOf(bitmap.width, bitmap.height)

    while (totalBytes + bytes > maxBytes) {
        if (!evictOldest()) break
    }

    buckets.getOrPut(key(bitmap.width, bitmap.height)) {
        ArrayDeque()
    }.addLast(bitmap)
    totalBytes += bytes
}

// Внутри evictOldest освобождаем старый буфер:
val bitmap = bucket.removeFirst()
bitmap.recycle()
```

Как это выглядит в коде: пул битмапов

НЕ ВЫДЕЛЯЙ НОВЫЙ БУФЕР, ПИШИ ПИКСЕЛИ ВОТ В ЭТОТ

```
val opts = BitmapFactory.Options().apply {
    inMutable = true
    inSampleSize = sampleSize
    inBitmap = pool.acquire(expectedW, expectedH)
}

val bitmap = decoder.decodeRegion(tileRect, opts)
```

```
private val buckets: LinkedHashMap<Long, ArrayDeque<Bitmap>> =
    LinkedHashMap(0, LOAD_FACTOR, true)

@Synchronized
fun acquire(width: Int, height: Int): Bitmap? {
    val bitmap = buckets[key(width, height)]?.removeFirstOrNull()
    ?: return null
    totalBytes -= bytesOf(width, height)
    return bitmap
}

@Synchronized
fun recycle(bitmap: Bitmap) {
    val bytes = bytesOf(bitmap.width, bitmap.height)

    while (totalBytes + bytes > maxBytes) {
        if (!evictOldest()) break
    }

    buckets.getOrPut(key(bitmap.width, bitmap.height)) {
        ArrayDeque()
    }.addLast(bitmap)
    totalBytes += bytes
}

// Внутри evictOldest освобождаем старый буфер:
val bitmap = bucket.removeFirst()
bitmap.recycle()
```

Как это выглядит в коде: пул битмапов

GC не нужен

буфер не уходит в heap, а возвращается
в пул через recycle()

НЕ ВЫДЕЛЯЙ НОВЫЙ БУФЕР, ПИШИ ПИКСЕЛИ ВОТ В ЭТОТ



```
val opts = BitmapFactory.Options().apply {  
    inMutable = true  
    inSampleSize = sampleSize  
    inBitmap = pool.acquire(expectedW, expectedH)  
}  
  
val bitmap = decoder.decodeRegion(tileRect, opts)
```



```
private val buckets: LinkedHashMap<Long, ArrayDeque<Bitmap>> =  
    LinkedHashMap(0, LOAD_FACTOR, true)  
  
@Synchronized  
fun acquire(width: Int, height: Int): Bitmap? {  
    val bitmap = buckets[key(width, height)]?.removeFirstOrNull()  
    ?: return null  
    totalBytes -= bytesOf(width, height)  
    return bitmap  
}  
  
@Synchronized  
fun recycle(bitmap: Bitmap) {  
    val bytes = bytesOf(bitmap.width, bitmap.height)  
  
    while (totalBytes + bytes > maxBytes) {  
        if (!evictOldest()) break  
    }  
  
    buckets.getOrPut(key(bitmap.width, bitmap.height)) {  
        ArrayDeque()  
    }.addLast(bitmap)  
    totalBytes += bytes  
}  
  
// Внутри evictOldest освобождаем старый буфер:  
val bitmap = bucket.removeFirst()  
bitmap.recycle()
```

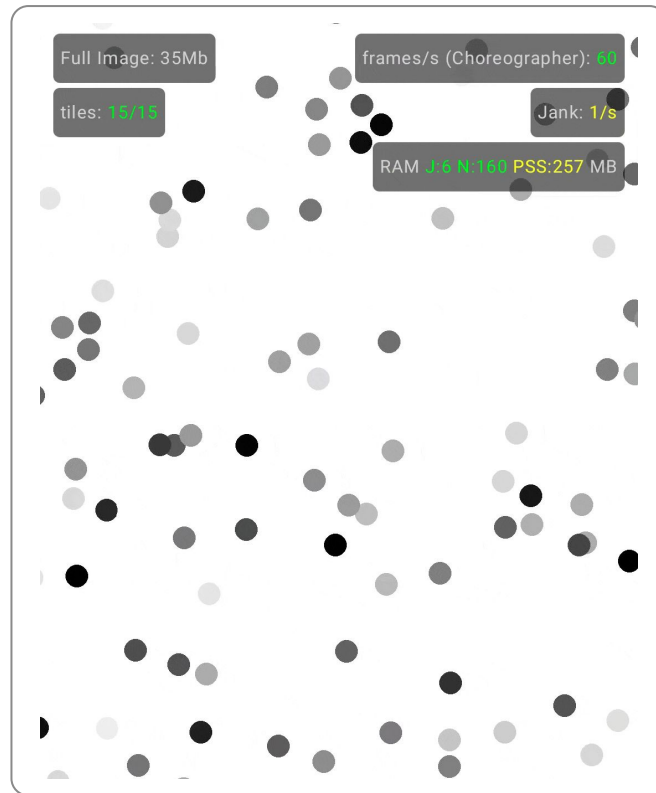
Загружаем в фоне — отменяем, если уехало

Пул битмапов

освободившуюся плитку
переиспользуем, а не аллоцируем заново

Асинхронная подгрузка с отменой

при быстром панировании половина запросов уже
неактуальна — отменяем



Как это выглядит в коде: асихронность

```
val job = scope.launch(dispatchers.io, start = CoroutineStart.LAZY) {
    try {
        ensureActive()
        val tile = decodeOrNull(rect, sampleSize) ?: return@launch

        try {
            ensureActive()
            pending[rect] = tile.image
            onTileDecoded()
        } catch (e: CancellationException) {
            tile.image.release()
            throw e
        }
    } finally {
        loadingJobs.remove(rect, coroutineContext[Job])
    }
}
loadingJobs[rect] = job
job.start()
```

```
// Кадровый цикл на main:
withFrameMillis {
    state.flushPendingTiles()
}

// Если тайл больше не нужен:
fun cancelTile(rect: TileRect) {
    loadingJobs.remove(rect)?.cancel()
    pending.remove(rect)?.release()
}
```

Как это выглядит в коде: асихронность

```
val job = scope.launch(dispatchers.io, start = CoroutineStart.LAZY) {
    try {
        ensureActive()
        val tile = decodeOrNull(rect, sampleSize) ?: return@launch

        try {
            ensureActive()
            pending[rect] = tile.image
            onTileDecoded()
        } catch (e: CancellationException) {
            tile.image.release()
            throw e
        }
    } finally {
        loadingJobs.remove(rect, coroutineContext[Job])
    }
}
loadingJobs[rect] = job
job.start()
```

Тайл = отдельная корутина в IO

```
// Кадровый цикл на main:
withFrameMillis {
    state.flushPendingTiles()
}

// Если тайл больше не нужен:
fun cancelTile(rect: TileRect) {
    loadingJobs.remove(rect)?.cancel()
    pending.remove(rect)?.release()
}
```

Как это выглядит в коде: асихронность

```
val job = scope.launch(dispatchers.io, start = CoroutineStart.LAZY) {
    try {
        ensureActive()
        val tile = decodeOrNull(rect, sampleSize) ?: return@launch

        try {
            ensureActive()
            pending[rect] = tile.image
            onTileDecoded()
        } catch (e: CancellationException) {
            tile.image.release()
            throw e
        }
    } finally {
        loadingJobs.remove(rect, coroutineContext[Job])
    }
}
loadingJobs[rect] = job
job.start()
```

Тайл = отдельная корутина в IO

ГОТОВЫЕ ТАЙЛЫ -> НА MAIN

```
// Кадровый цикл на main:
withFrameMillis {
    state.flushPendingTiles()
}

// Если тайл больше не нужен:
fun cancelTile(rect: TileRect) {
    loadingJobs.remove(rect)?.cancel()
    pending.remove(rect)?.release()
}
```

Как это выглядит в коде: асихронность

```
val job = scope.launch(dispatchers.io, start = CoroutineStart.LAZY) {
    try {
        ensureActive()
        val tile = decodeOrNull(rect, sampleSize) ?: return@launch

        try {
            ensureActive()
            pending[rect] = tile.image
            onTileDecoded()
        } catch (e: CancellationException) {
            tile.image.release()
            throw e
        }
    } finally {
        loadingJobs.remove(rect, coroutineContext[Job])
    }
}
loadingJobs[rect] = job
job.start()
```

Тайл = отдельная корутина в IO

ГОТОВЫЕ ТАЙЛЫ -> НА MAIN

```
// Кадровый цикл на main:
withFrameMillis {
    state.flushPendingTiles()
}

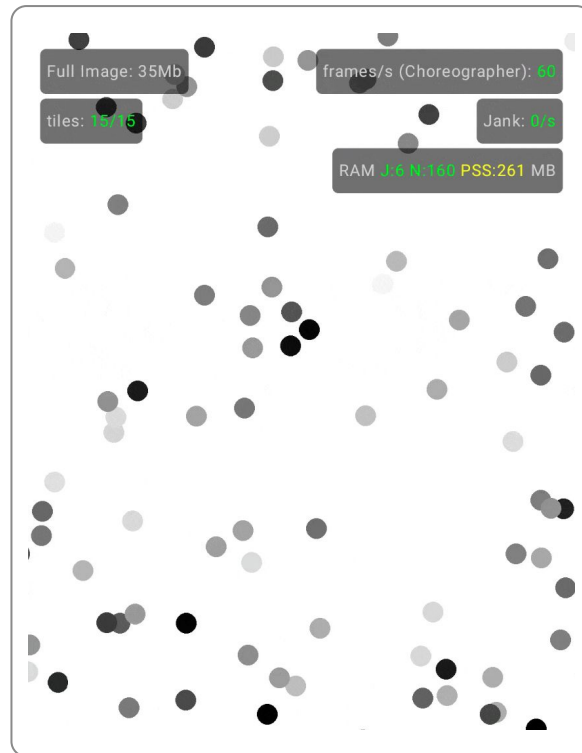
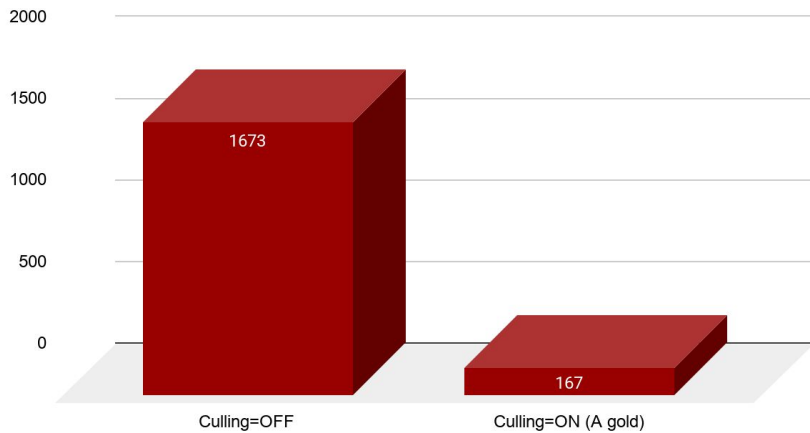
// Если тайл больше не нужен:
fun cancelTile(rect: TileRect) {
    loadingJobs.remove(rect)?.cancel()
    pending.remove(rect)?.release()
}
```

ТАЙЛ УШЕЛ -> КОРУТИНУ ОТМЕНИЛИ

Картинка, которая раньше падала

Открываем то, что крашилось — теперь работает.

RAM NATIVE



Враг №2: стоимость кадра

Как движение превращается в жест

SURF . RU



Как движение превращается в жест

Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Как движение превращается в жест

`pointerInput`

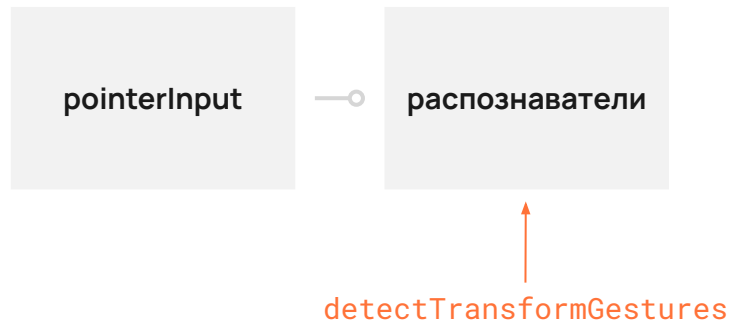
Слушаем касания → навешиваем `pan/zoom` → добавляем инерцию.

Как движение превращается в жест



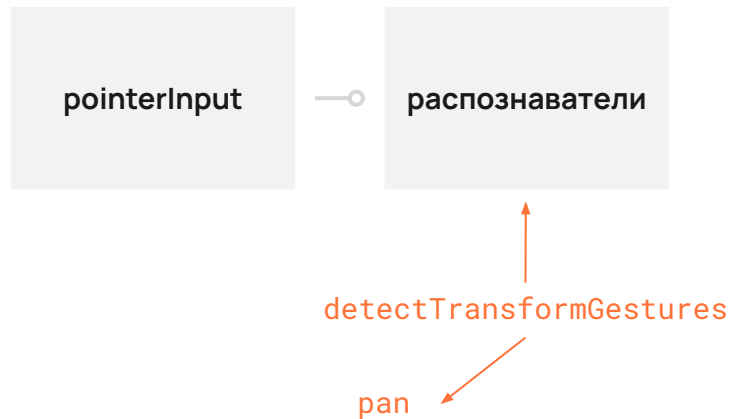
Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Как движение превращается в жест



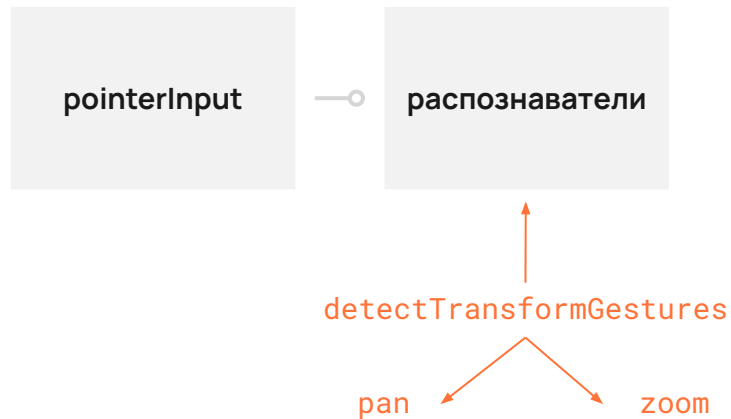
Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Как движение превращается в жест



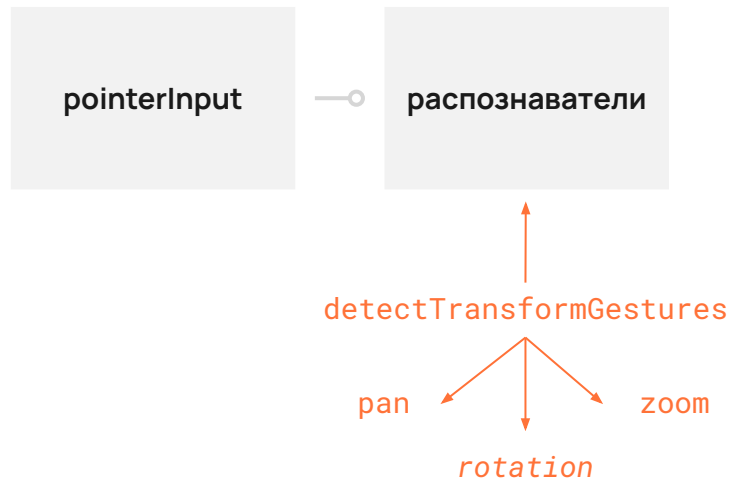
Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Как движение превращается в жест



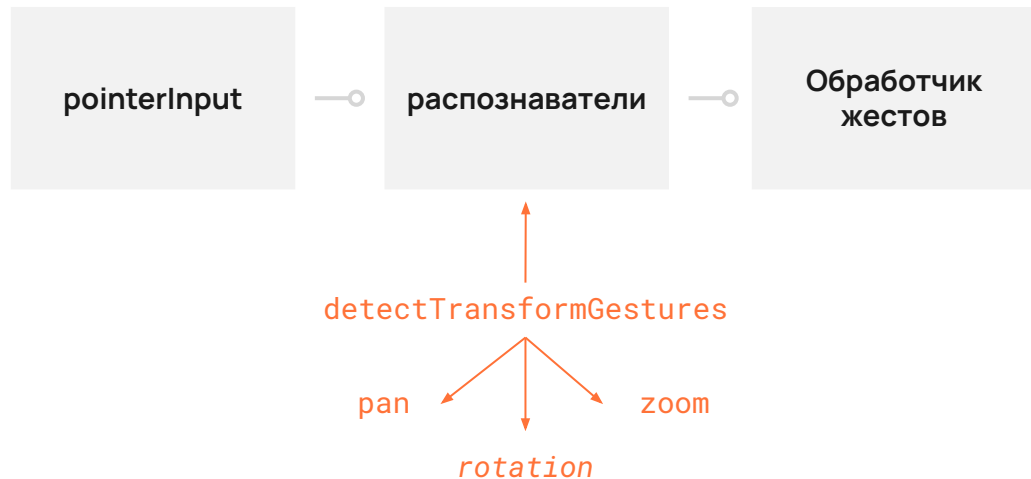
Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Как движение превращается в жест



Слушаем касания → навешиваем `pan/zoom` → добавляем инерцию.

Как движение превращается в жест



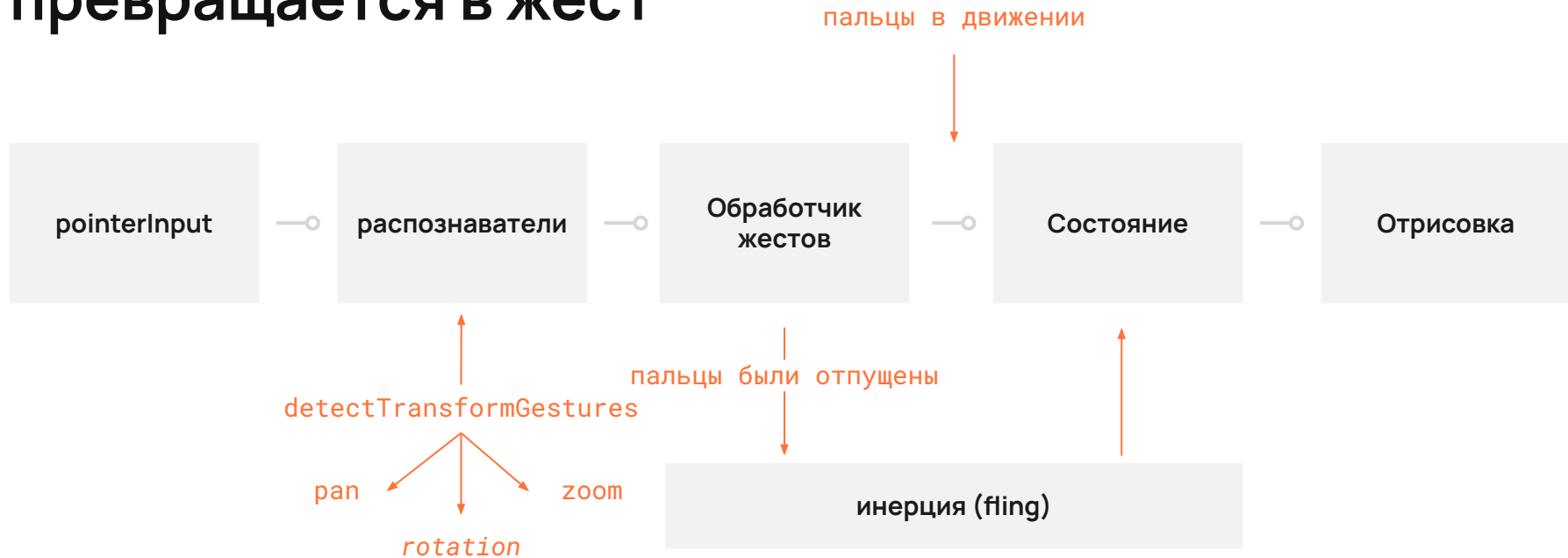
Слушаем касания → навешиваем `pan/zoom` → добавляем инерцию.

Как движение превращается в жест



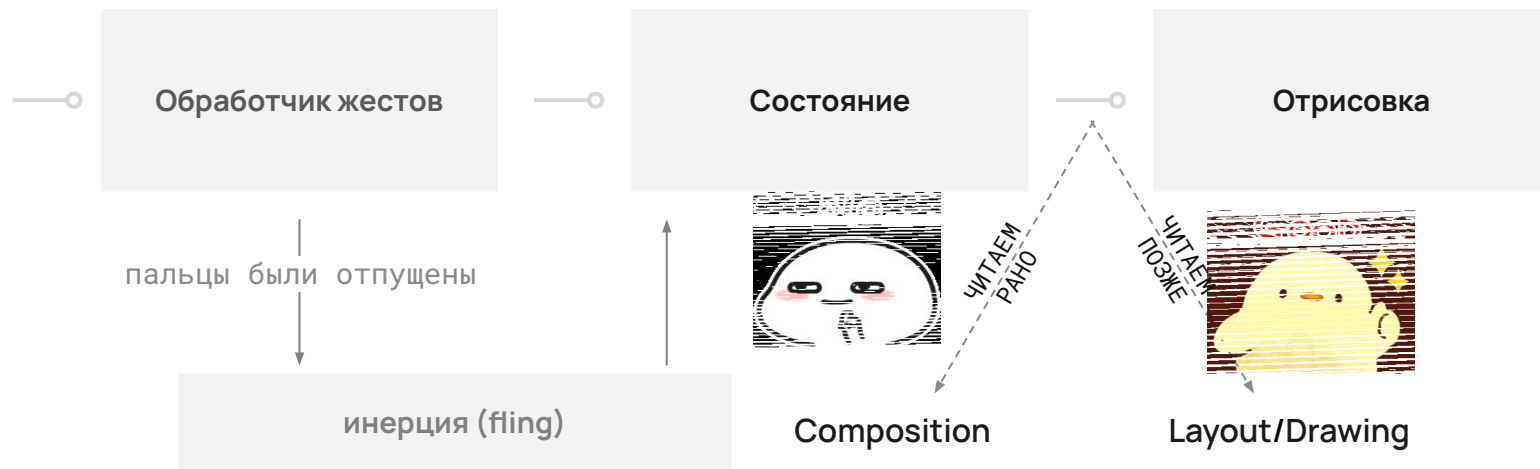
Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Как движение превращается в жест



Слушаем касания → навешиваем pan/zoom → добавляем инерцию.

Стоимость рекомпозиций



Почему рекомозиции — это важно?

Обычный телефон



Слабое устройство

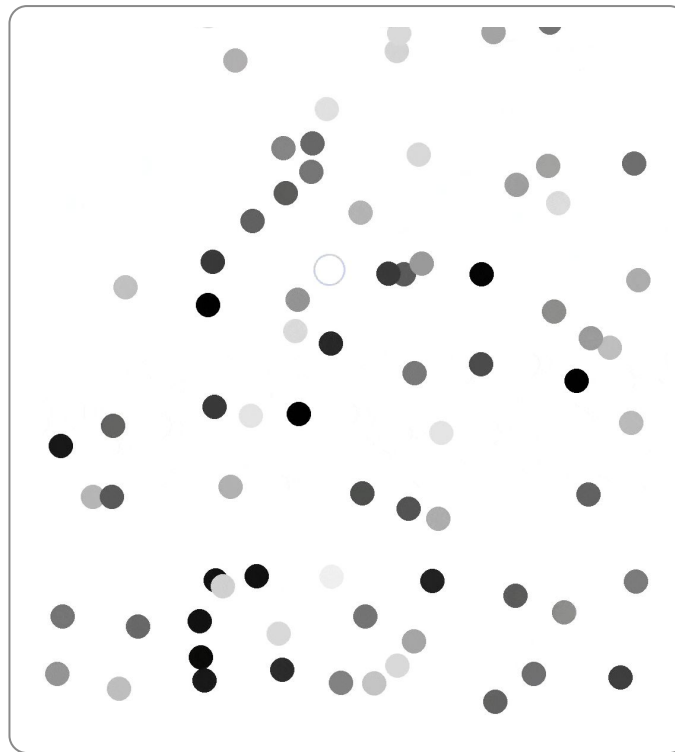
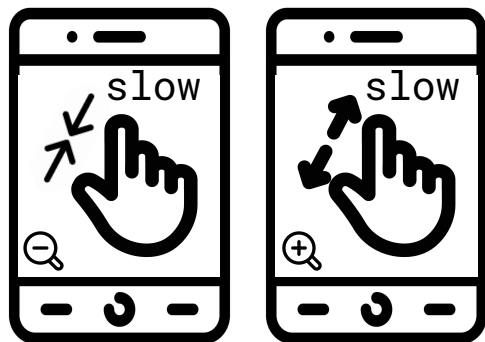
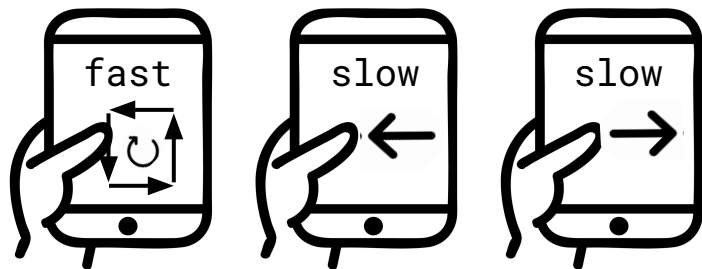
Пересмотри подход



```
// ❌ Параметрическая форма: переменная читается в фазе сборки
//   → пересборка на каждое движение пальца
Modifier.graphicsLayer(
    scaleX = state.scale,           // чтение в фазе сборки
    translationX = state.offset.x  // !!! Compose видит зависимость
)

// ✅ Лямбда-форма: переменная читается в фазе отрисовки
//   → фаза сборки не запускается вообще
Modifier.graphicsLayer {
    scaleX = state.scale           // чтение в фазе отрисовки
    translationX = state.offset.x  // Compose не видит зависимость)))
}
```

Сценарий: GraphicsLayer=OFF



Метрики

Метрика	GL=ON (A)	GL=OFF	Δ
Compose:recompose	14	526	×37.6
Choreographer# doFrame	1.66 мс	1.99 мс	+20%
App Deadline Missed	9	4	-5

Метрики: PERFETTO

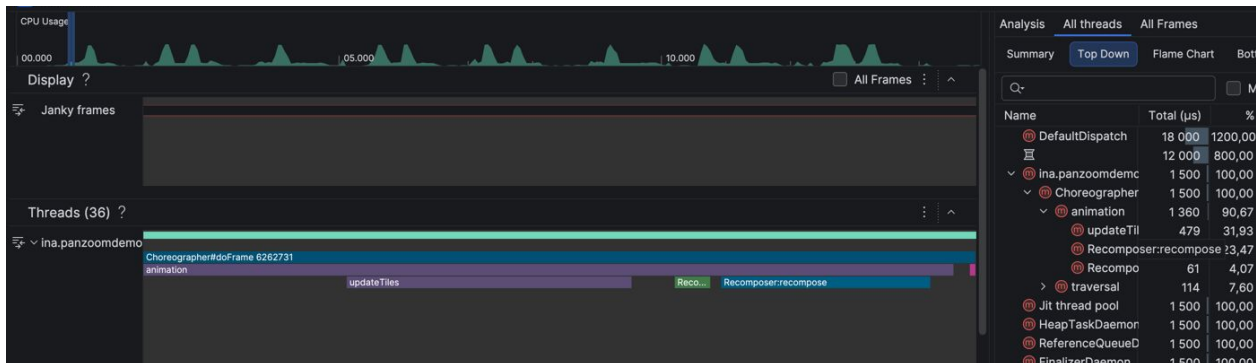
GL=ON

doFrame

→ animation

→ Recomposer:recompose

сразу на GPU



GL=OFF

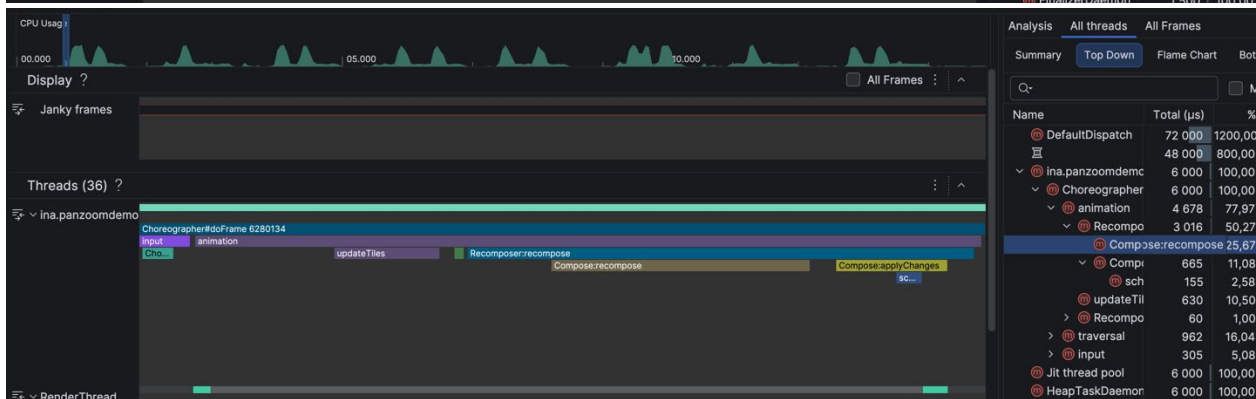
doFrame

→ animation

→ Recomposer:recompose

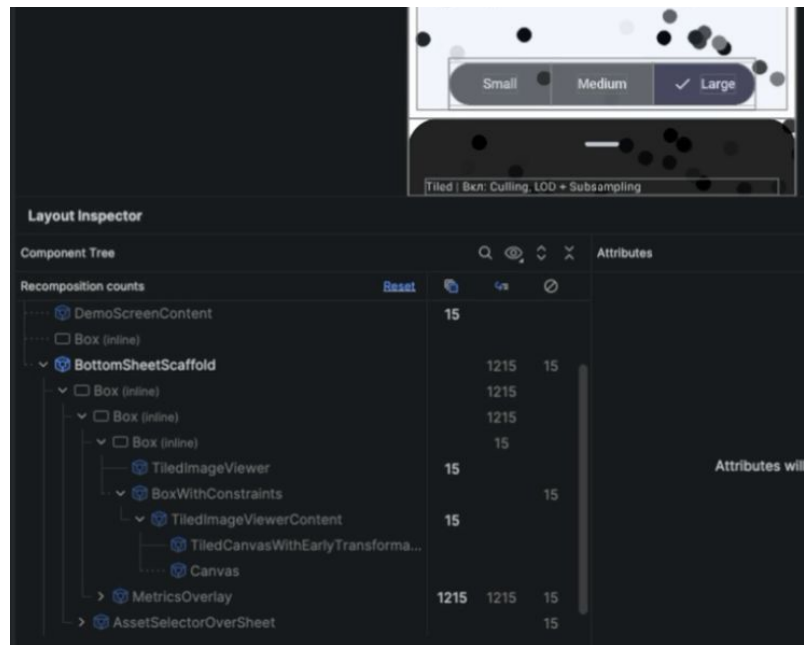
→ Compose:recompose

каждый кадр

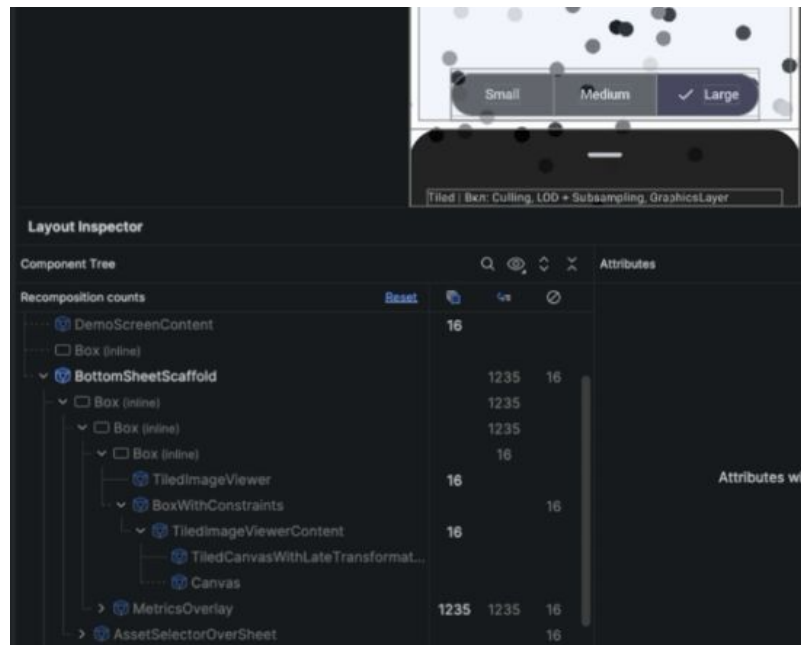


Layout Inspector

GL=OFF



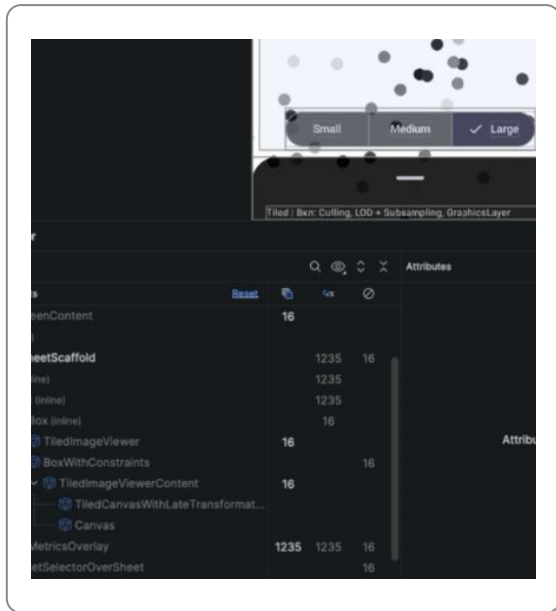
GL=ON



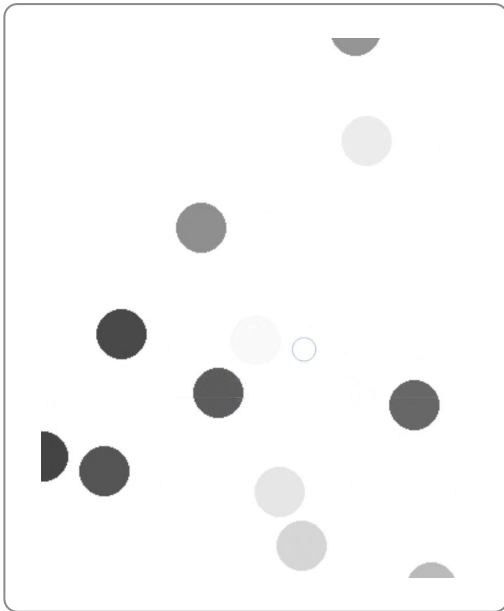
Финальное сражение: 60 fps на сцене

Посмотрим на силу каждого оружия

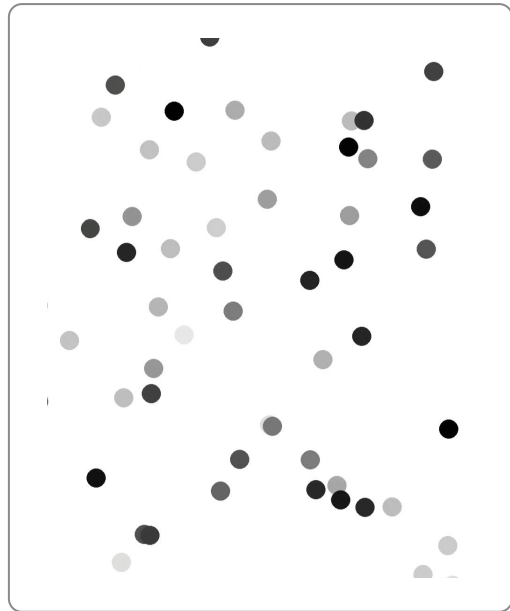
graphicsLayer



LOD



culling



Самые важные вопросы

№1

ВЛЕЗАЕТ ЛИ В ПАМЯТЬ?

№2

КАКАЯ ЦЕНА КАЖДОГО КАДРА?

Знаешь врага = знаешь, в какую фазу Compose целиться.

Метрики

Оптимизация	WITH	WITHOUT	Δ
Culling	RAM N 162 МБ	RAM N 1 672 МБ	×10.3
LOD	RAM N 206 МБ	RAM N 664 МБ	×3.8
TilePool	Pool hit-rate 98.8%	Pool hit-rate 1.2%	+97.6 п.п.
graphicsLayer	Recompose 14	Recompose 526	×37.6

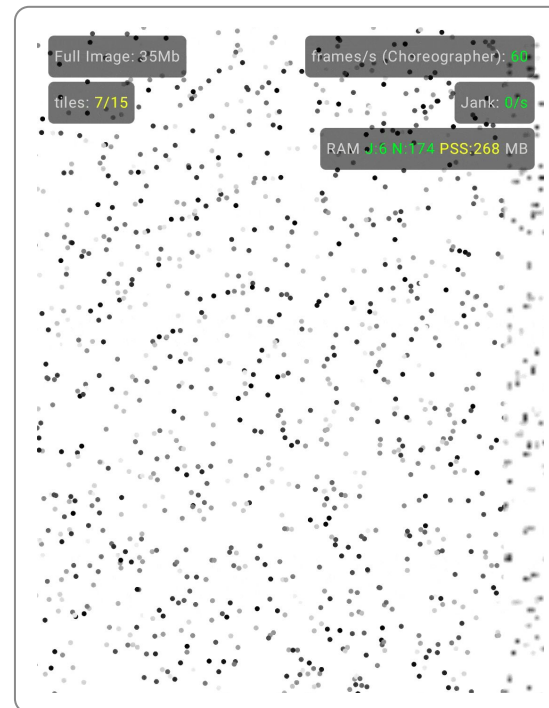
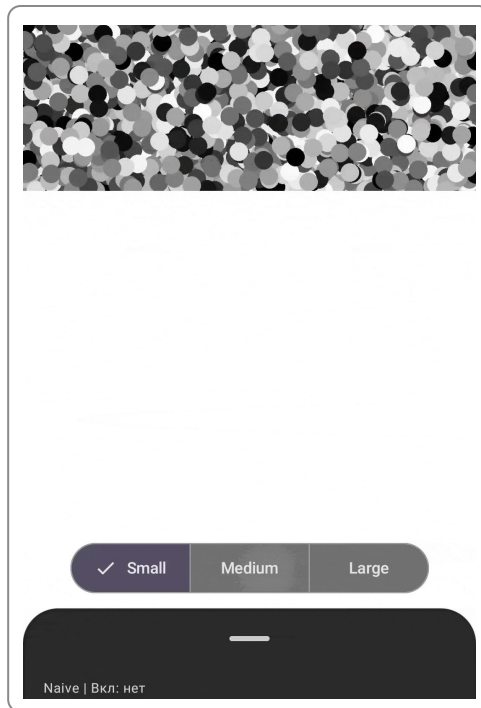
Лайфхаки

Враг	Оружие	API	Что делает
Память	Региональное декодирование	<code>BitmapRegionDecoder</code> + culling	Берет только видимую область, не декодируя картинку целиком
	Субсэмплинг	<code>inSampleSize</code> + LOD	Грузит в нужном разрешении, грубее при отдалении
	Переиспользование	<code>inBitmap</code> · пул	Сокращает аллокации → снижает нагрузку на GC во время жеста
Стоимость кадра	Поздняя трансформация	<code>graphicsLayer { }</code>	Применяет pan/zoom как матрицу слоя на GPU, без лишних рекомпозиций.

Было → стало



ТА ЖЕ КАРТИНКА,
ТОТ ЖЕ ТЕЛЕФОН —
ИЗ МЕДИЦИНСКОЙ
ВИЗУАЛИЗАЦИИ
В ОБЫЧНЫЙ
МОБИЛЬНЫЙ UI.





Спасибо!

SURF

Дарья Городнянская

ANDROID-РАЗРАБОТЧИК В SURF



LinkedIn



@SHEVIKINA

КОД ДЕМО

