Postmortem: Vector Unit's Riptide GP

PRINT

By Matt Small

[*What does it take to deliver a console-like experience on Android? Vector Unit moved from XBLA title Hydro Thunder Hurricane to Riptide GP and here shares the ups and downs of creating a high-powered water-based racing game for Google's mobile platform.*]

In June of 2010, the team at Vector Unit was feeling cautiously optimistic. We'd just wrapped up development on our first game, *Hydro Thunder Hurricane,* on time and on budget, and were preparing to launch it in one of XBLA's coveted Summer of Arcade slots. Early reviews were good, and sales projections indicated we'd be in royalties before the end of the year.

We still had a little income coming in via the *Tempest Pack* DLC for *Hurricane;* it was enough to keep me and one contract artist busy through the summer. Since it didn't require a lot of engineering work, Ralf Knoesel, my co-founder and our lead programmer, took a contract at EA Visceral Studios to work on *Dead Space 2* while we continued to pitch Microsoft and other publishers on new projects.

We had some great new concepts that we thought were sure to get picked up by somebody, given what we thought was a pretty solid freshman effort, but nothing had stuck. For one thing, publishers tended to think of us as a "racing game studio", and with some recent high-profile flops in the racing genre, there wasn't a huge appetite for new racing games.

Hurricane launched in July. Reviews were solid and it was well received by players, but sales -- although good by general XBLA standards -- weren't as off-the-charts as we'd hoped. It soon became clear that self-funding a second title would not be an option, at least not any time soon. In October I wrapped up work on the *Tempest Pack*, Ralf was still contracting at EA, and we still had no clear second project on the horizon. The question of what to do next began to feel urgent.

Then in early November we got a surprise call... from NVIDIA. The chip maker was moving into the Android phone and tablet markets with its dual-core Tegra 2 processor, and it was looking for games to showcase the new hardware's capabilities. Someone there had played *Hurricane*, and wanted to know if we could build something similar for them.

The deal wasn't exactly what we'd been looking for, and the challenges sounded formidable. Neither of us had ever worked on a mobile game. We had no idea what the Tegra 2 hardware was really capable of, or if it would even be possible to pull off the kind of water physics and effects we'd developed for *Hurricane* on the 360. The budget was tight -- we'd have to scrounge up investment for most of the development costs and cover the difference ourselves. And the schedule was aggressive -- first playable by February, finished game by May.

An early shot from Alpha City, right after the MWC demo in March.

On the bright side, we'd planned on porting our engine to Android at some point, and this would be an opportunity to create one of the first games developed specifically for high-end tablets and phones. It would be hard work, but it sounded like fun. And hey -- work is work.

So throughout December we hammered out deal terms, and with NVIDIA's help we managed to cobble together funding (more on this later). Knowing what was coming, we allowed ourselves a little break for the holidays, and at the start of January 2011 we dove headfirst into our studio's second game -- a futuristic jet ski racer called *Riptide GP.*

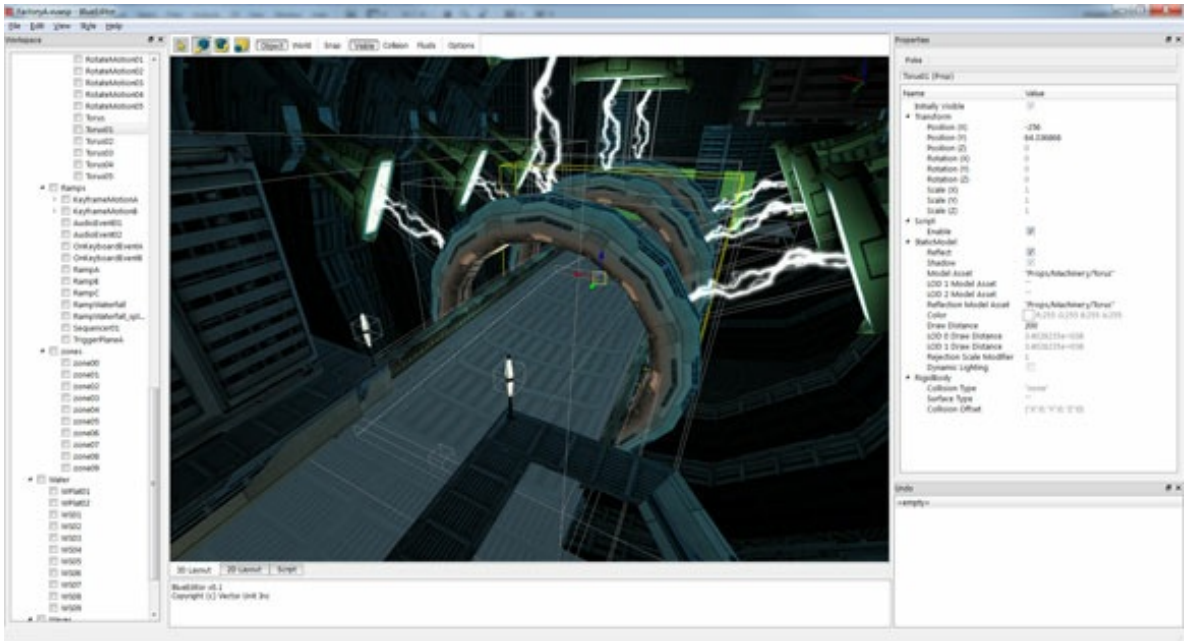
What Went Right

1. Standing on the Shoulders of *Hurricane*

It simply would not have been possible to make *Riptide GP* in the time we had without *Hydro Thunder Hurricane.*

Vector Unit owns all the technology we developed for *Hurricane*, so when we began work on *Riptide GP* in early January, we were able to hit the ground running with a full-featured, cross-platform engine that allowed me to begin content development on the PC as Ralf worked on porting the low-level stuff over to Android's NDK.

As I began prototyping our first vertical slice, Ralf quickly found that the Tegra 2's CPU was fully up to the task of handling the calc-intensive fluid dynamics simulation he'd used on *Hurricane*. He was able to port over all of our water, wave, and Bullet physics code as written from the 360 with little or no modification. This isn't to say there weren't challenges on the code side (see "What Went Wrong"). But, generally speaking, the cross-platform groundwork laid on *Hurricane* paid off, and the port to Android went relatively smoothly.



Major environment components were built and assembled in Maya, and then exported and brought into the Vector Engine game editor. We then add in PFX, animated objects, AI pathing, and water surfaces (click for full size).

We also had the benefit of two years of water-racing design experience, and two years of learning from our mistakes. We knew from the start there were a number of things we could and couldn't do, and a few we wanted to change.

For example, we had never been completely satisfied with the water sim in *Hurricane*; it was actually too realistic. When you watch high speed offshore racing, the boats don't sink into the water -- at 100mph, the water might as well be asphalt. But in a game, you want that feel of softness.

In *Hurricane*, we started with a realistic water sim, and sort of backwards-engineered the softness to get the boats to sink more into the water. For *Riptide GP*, Ralf went into the base simulation and modified the underlying buoyancy and resistance calculations to give the water a much more pleasant, soft feeling, while still maintaining realistic handling characteristics.

With less than five months to make the game, we never would have had the time for this type of discovery and iteration if we had been starting from scratch. We came off *Hurricane* with a pretty good feel for what worked, and what didn't, in a water-based arcade racer, and were able to apply all of that experience to *Riptide GP.*

---

2. Ruthless Design Scoping

We started *Riptide GP* with four pretty straightforward underlying design goals.

First and foremost, as a showpiece for Tegra 2 it had to be technologically and visually impressive. Second, it was aimed at mobile platforms, so we wanted an accessible, easy-to-pick up game mechanics and short, two to three minute play sessions.

Third, we wanted the game to have enough replay value to justify a "premium" price point; not that we have anything against the Freemium or 99 cent app models, but we presumed that there wouldn't be enough of these new devices on the market to support a high-volume revenue model.

And finally, of course, we had to be able to make the game with two people in just four and half short months...

primarily because we wanted to be one of the first titles available on the emerging platform.

Our second set of goals -- simplicity, brevity -- dovetailed nicely with the tight schedule. But striving for console quality visuals and scope within that time was a challenge. To create sufficient content -- particularly for environments -- we built everything modularly, instancing and repurposing geometry and textures wherever possible.

This reuse had the added benefit of keeping our pack size small; in the end we came in under Google's 50mb limit without requiring an additional data download, and our load times are quick enough that we didn't need a loading bar.

In our initial design specs and throughout development, we were ruthless about stripping away any elements that might distract from our core goals. Many otherwise desirable features -- online multiplayer, customizable characters and vehicles -- were left on the cutting room floor.

This stripped-down focus on core gameplay allowed us to make fast progress. After about two months of development, the game was coming together and the controls felt great. But the overall experience still felt a bit shallow, and we decided to reintroduce a mechanic we'd cut early on. We added the ability for players to earn extra boost by performing stunts with different combos of thumb-swipes on the screen when in mid-air.

Stunting provided the extra kick (ahem) that the game needed. The gestural stunting was fun, and tying successful stunts to boost provided the risk-reward tension the game had been lacking.

3. Easy Does It

Riptide GP was our first experience designing and building a mobile game, after years working on console titles. We didn't want the game to be casual per se, but we felt that the game should be more accommodating to casual play styles, both in terms of difficulty and control mechanics.

Racing games on console -- even arcade racers like *Hydro Thunder Hurricane* -- tend to be very technically challenging. Much of the fun and reward comes from mastering the vehicle controls and physics, carefully managing speed, and picking out the perfect racing line through a variety of corner angles and track widths. The conventions of mobile racing games, on the other hand, with accelerometer steering and pedal-to-the-metal auto-acceleration, don't exactly lend themselves to nitpicky perfection.

Rather than struggle against this, we decided to embrace it. Our vehicle specs and track layouts were all designed to be driven at full speed, with braking only really necessary when the player takes a corner wrong. We wanted the game to feel fast and flowing, and we looked to the *Wipeout* series for inspiration on track layout. There are very few irregular or hard corners where players can catch edges or get stuck -- when you hit a wall you slow down, but you keep moving forward.

As a result, *Riptide GP* is incredibly easy to pick up and play. On most tracks, you can almost let the game play itself -- you'll finish in last place, but you'll finish. Depth and challenge come from learning how to keep your speed up in the corners, interacting with the waves and other riders' wakes, and knowing when it's safe to pull off a stunt for that precious extra bit of boost.

At first, I was concerned that the game might be too easy. But we playtested the hell out of it with friends and anyone else we could get our hands on, and by providing three different difficulty levels, we found we could accommodate both casual and hardcore play styles without overly frustrating the one or boring the other.

A previs concept rendered in the *Hydro Thunder* engine on the 360, before we ported to Android. We had to lose normal maps and fogging, but otherwise managed to come pretty close to our visual target.

4. First!

Jumping on board the Tegra 2 train just as it was pulling out of the station proved to be a big win for us.

From a production standpoint, it meant a manageable target. Unlike on iOS, where you have this sort of benign dictatorship mandating consistency in hardware and software updates, hardware and OS fragmentation is a serious challenge for Android developers.

By focusing on Tegra, we had a relatively consistent performance and tech standard to steer towards. And because most of these devices are new, we only had to worry about a few different versions of the Android OS. All of this vastly simplified our requirements for compatibility testing... which is good, because we didn't really have any time in the schedule for it.

Additionally, because we were exclusively targeting its platform, NVIDIA was incredibly helpful. The company provided development hardware and soon-to-be-released devices to test on. Its tech team helped evaluate our builds and provided critical optimization feedback. And the marketing team introduced us to many of the carriers and manufacturers launching new devices, who of course were looking for new games to differentiate their products. Their support helped generate buzz and gave us much needed exposure.

5. Fairness

Okay, we all know this, but I need to say it: the traditional game financing model, with its publisher advances and royalty recoupment, sucks.

One of the best things about *Riptide GP* is that we managed to skirt the traditional system and fund the game with a mixture of our own cash and third-party investments. As I mentioned in the Introduction, NVIDIA helped us with this. The company introduced us to Flashman Capital, a company in San Francisco that helps small studios with development financing. Flashman agreed to pony up some of the cash we needed, in exchange for a revenue share; we scrounged up some other investment and covered the difference out of our own pockets.

There were two key benefits to the deal we managed to pull together. First, we got to keep the IP, which among other things means that, after a brief exclusivity period on Tegra, we can take the game onto whatever other platforms we want.

Even better, we and the other investors all share in the game's revenue. With the traditional publisher model, the developer doesn't see a penny until the publisher's initial investment gets paid back out of the developer's royalty share. By the time that happens, if it ever does, the publisher has made back several times their initial investment. In the revenue share model, all the investors -- including us -- share in the proceeds from day one, more or less according to respective investment levels.

As a result, Vector Unit is much more invested in the game's success. We're motivated to promote and nurture the game and its community, because we see a direct benefit from every additional sale. And just generally, we're happier, because the whole thing feels strangely... fair.

What Went Wrong

1. Great Expectations

The first few weeks of January, when we were porting our Vector Engine over to run on the Tegra 2 dev hardware, were a bit of a nailbiter.

We had signed up to deliver an in-engine demo for NVIDIA to show at Mobile World Congress in early February. We didn't know what to expect from the hardware in terms of real performance, but we had high hopes. All we had to go on were the paper specs and the expansive claims from NVIDIA and early-adopter developers that Tegra 2 could provide a "console quality" experience on mobile.

Well, as is generally true with performance claims for new hardware, the reality was a bit more complex.

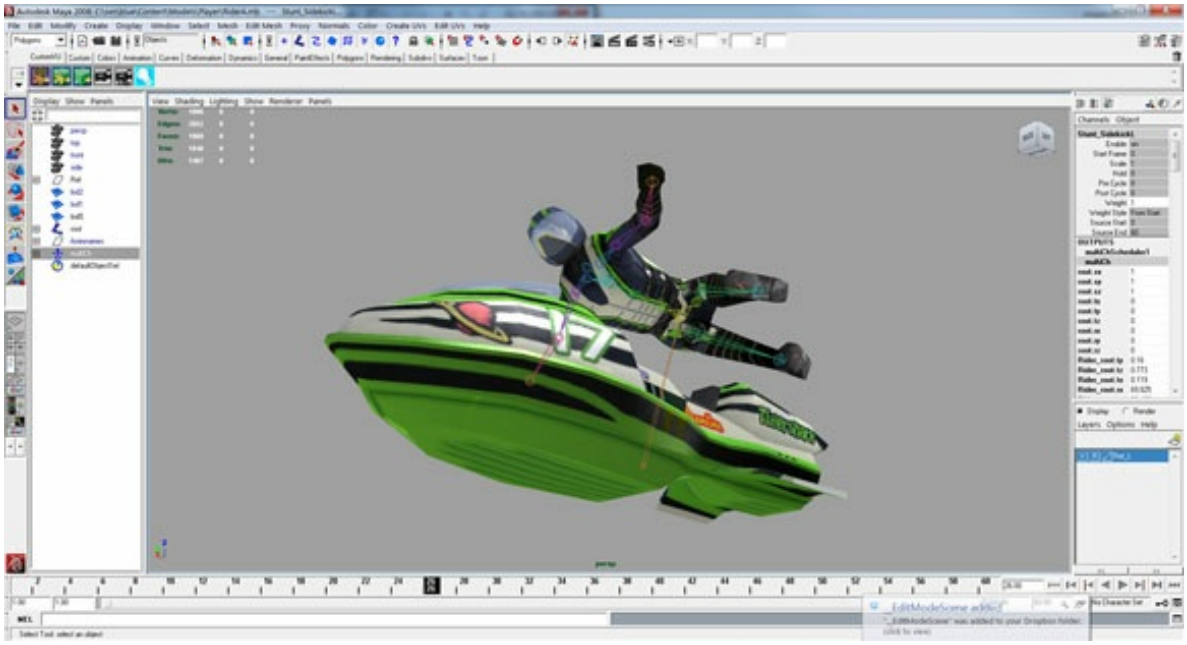
As I mentioned earlier, porting our 360/PC cross platform engine over to the Android NDK went smoothly. By the end of the second week, Ralf had my rough prototype level up and running on the Tegra 2 dev kit, and in some respects the Tegra 2 hardware exceeded our expectations.

The CPU was amazing -- it ate up our water sim and fluids calculations, munched happily on a straight implementation of Bullet Physics, chewed up my high-poly models, and asked us for more.

The GPU was another story. We were extremely fill-rate bound, which is particularly a challenge in a game where almost half of the screen space is covered by a giant translucent water mesh. We quickly realized that the complex pixel shaders we'd relied on so heavily on the 360 just wouldn't allow us the framerates we needed for a fast-paced racing game.

We streamlined our shaders, moved every per-pixel process we could to the per-vertex level, and switched to low-precision calculations wherever possible. We stripped normal mapped ripples off the water and increased mesh density to maintain the complexity we needed to break up reflections. I dusted off my old skool artist toolkit, baking lighting into vertex colors, painting lighting detail into diffuse textures, and ruthlessly managing texture and mesh variation to keep the GPU pipe as open as possible.

NVIDIA provided a ton of helpful performance analysis throughout this process, and in the end, we were able to deliver that first in-game MWC demo with a level of visual quality that, at least to the untrained eye, really does look comparable to something you might see on a modern PC or console. The rate of technological advance on mobile platforms these days is staggering, and I expect that within a few years even savvy gamers will be hard pressed to tell you whether an in-game screenshot was rendered on a high-end console or on a phone.



Modeling, animation, and shader adjustment was all done in Maya 2008 (click for full size).

2. Creative Reuse, or the Lack Thereof

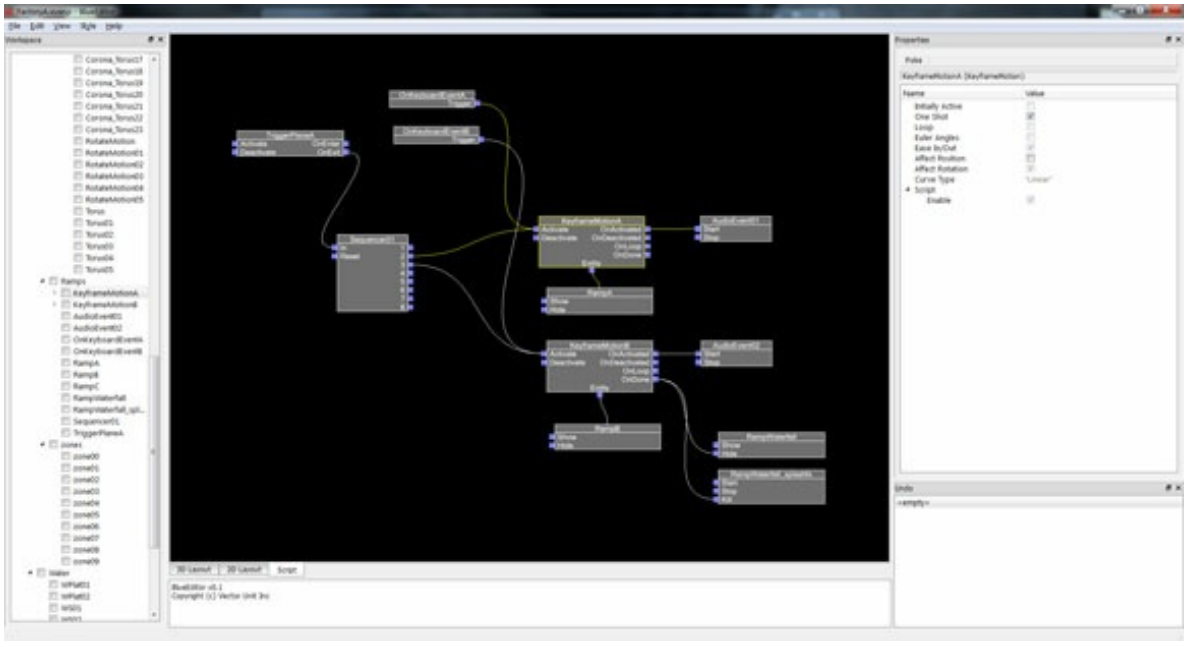
From the start we knew that content creation would be critical path in *Riptide GP's* development. Unlike the code, which we owned outright, we weren't able to reuse any content -- models, texture libraries, anything -- from *Hurricane*, so I had to start from scratch.

When I first broke down my 4.5 month art schedule, it looked pretty scary. I had about four weeks for the jet skis, characters and animations, 10 weeks for environments, and five weeks for everything else, including UI, particle effects, polish, optimization, and whatever unforeseen gotchas might lay in wait.

The environment schedule was the hardest. On average I had about six to seven days to finish each of our six main race tracks, with another day or two allocated for creating each reverse variant, which I wanted to be at least a little different, with unique lighting and water features.

The only way this would be possible was through conscientious reuse and instancing, and to some extent I was able to make this work -- the tracks were built from instanced segments, and I looked for every opportunity to reuse and recycle textures and geometry from track to track. But I couldn't quite let go of my desire to try and make each track feel different, and I didn't reuse as many assets as I could have.

In the end, I got it all done the old fashioned way, through hard work and long hours. Overall I didn't mind; it was creative work, and as I mentioned above, we were energized and heavily invested in delivering a quality game. I was still polishing and tweaking details the night before we released *Riptide GP* on the Android Market in May. But I will say I was pretty relieved when we finally hit that Publish button.



All game scripting is built up using visual scripting components. This sequence controls the animated ramps in "Grindhouse" (click for full size).

3. Small Team Challenges

We didn't start out intending make *Riptide GP* with just the two of us. I'd thought I might contract out a month or two of artwork, and we had some budget allocated for things like sound design and music.

But because we were partially funding *Riptide GP* out of our own pockets, we were powerfully motivated to keep costs down. And because our schedule was so tight, we felt we couldn't afford the time it would take to manage outsourcing.

So we just went full steam ahead. My schedule was pretty much filled from start to finish with art tasks, but as it turned out, Ralf's programming schedule started to open up once he got past the graphics engine optimizations. He ended up taking on a ton of miscellaneous content tasks, including sound design and music curating (we sourced most of it from online stock sites like Audiomicro), screens flow scripting, and even some particle effects.

The problem as always with a very small team -- in our case, a very, very small team -- is that there is virtually no wiggle room for unforeseen variables. We went from three week Scrum sprints at the start of the project, to two week sprints, to one week, and towards the very end we tossed Scrum completely and just went with prioritized task lists, squeezing as much as we could into the time we had. As hard as it was, it was creative, rewarding work, and somehow we managed to get it all done on time. But the process wasn't always pretty.

---

4. Yarrr!

Piracy is a problem in mobile that nobody likes to talk about. It's kind of embarrassing for the platform holders, and I think most developers feel it's in their best interest give the issue as little publicity as possible. But it is a bona fide issue, particularly for a game like *Riptide GP t*hat monetizes strictly off base game sales and not through in-app purchases.

We took Google's advice and implemented their Android Market license-checking. Ralf obscured the license code wherever possible, looking for ways to outmaneuver at least the automatic bots that strip licenses on new executables and post them automatically to warez sites. We knew that if the hacker community really wanted to they could find a

workaround, but at least they'd have to work for it.

Well, they didn't have to work that hard. When the game first launched, we were elated to see posts in the warez forums indicating that the automatic license stripping wasn't working. But within 24 hours, a playable cracked version of *Riptide GP* showed up.

According to our analytics, as of this writing the ratio of pirated copies to legal copies out there in the world is about 9 to 1. Our assumption -- our hope at least -- is that the majority of pirates probably wouldn't have bought the game anyway, so the numbers don't precisely translate into lost sales.

The annoying thing, though, is that for the players who buy the game legally, Google's license check can be a little persnickety, particularly if the user has a spotty mobile internet connection. We don't get very many tech support emails -- I'm happy to say the game itself is pretty stable -- but the majority of emails we do get have to do with failed license checks. They can be answered and resolved with a simple form letter, but it's time consuming, and annoying, and it makes us wonder whether it was worth implementing the license check at all.

5. Where is the Love?

Coming off *Hydro Thunder Hurricane* on XBLA, we thought we had the whole press-promotion thing figured out: You send out some press releases with screen shots and video for preview coverage, send out review codes for reviews, and bam, you're done! Of course we were hugely helped by the fact that *Hydro Thunder* was a known franchise, but from what we could see, the same strategy more or less worked for original indie games on console as well.

Not so with mobile games -- it's particularly hard to get coverage for Android games, and even more challenging for Tegra-exclusive games. Few of the major gaming sites even offer dedicated mobile coverage, and those that do mostly just cover the biggest iPhone/iPad titles. As of this writing, you won't find a single Android game listed on Metacritic. And outside of the mainstream gaming press, the major tech outlets like CNet and Wired focus primarily on hardware and non-gaming apps.

For *Riptide GP*, we partnered with marketing group Flashman Agency to try and make the biggest splash we could. We worked with NVIDIA and carrier/hardware partners to showcase the game at press events and trade shows like MWC, GDC, and E3. We sent out press releases, trailers, screenshots and other assets to all the major gaming and tech outlets.

Despite this, most of our efforts announcing *Riptide GP* to the world were met with... crickets. Well, that's not entirely fair. Joystiq covered us, and we got a decent amount of traction and praise from many smaller to mid-sized Android and mobile gaming sites and blogs. But the major gaming and tech sites were simply not interested.

For those sites that did cover us, our challenge became one of infrastructure. The Android Market doesn't have any review code mechanism, so there is no easy way to send a copy of your game to the press to review. One additional challenge was that not a lot of sites even had Tegra hardware they could play the game on. We ended up getting a bunch of loaner phones and tablets that we preinstalled the game on and sent out to press ourselves.

I understand the lack of interest from major outlets is partly a question of platform market share. But I believe it's also due to prejudice on the part of the mainstream gaming press -- mobile games still have a reputation for being shallow, throw-away experiences, unlike "real games" developed for PC and console.

But the high-end mobile market is growing much faster than the console market, and there is a whole new generation of mobile games coming out which offer experiences every bit as compelling as what you can find on traditional gaming platforms. Consumers already know this. The question is: when will the gaming press catch up?

A shot from the final game. The new billboard foliage system allowed us to mix a few naturalistic tracks in with the gritty industrial ones.

Epilogue

Riptide GP launched in the Android Market on May 20. Sales have been pretty good, at least relative to other Tegra-exclusive games. Better still our user reviews and comments have been really encouraging -- the game is holding an average user score of about 4.8/5. An iOS version is in the works, and we're considering other platforms as well. In addition, there's always the possibility of downloadable content and other offshoots.

Most importantly, this experience has given us a taste of success in a huge new market. We've always wanted to keep our team small, and assumed that eventually that would mean shifting our focus from console to mobile -- we'd just thought that move might be a few years down the road.

This isn't to say we've given up on console development. We see it as part of the mix. Consoles still deliver superior technical punch, and the increased scope and budgets for funded titles offer greater stability. Also, console gamers still demand -- and allow us to explore -- a little more depth and sophistication in game mechanics than we might currently do on mobile.

Still, the mobile gaming market offers us a chance to experiment with smaller-scale projects, to put many small bets on the table as opposed to a few big ones. The games are fun to work on and easy to wrap your head around. Best of all, we actually might be able to self-fund a few and potentially reap the full reward of our work.

I hesitate to say we've got this small-studio thing figured out. But at least for the foreseeable future we have enough work to remain stable. And right now stability feels really good.

Data Box

Title: *Riptide GP*

Released: May 20, 2011

Platform: Android/Tegra2

Genre: Racing

Engine: Vector Engine

Team size: 2

Development time: 11 man-months

Budget: $100k

3rd Party Tools: FMOD Sound System, Bullet Physics (for collisions and ragdoll)

Development Tools: Maya 2008, Photoshop CS4, Microsoft Visual Studio 2010, Subversion

Links

Web: www.riptidegp.com

Android Market: https://market.android.com/details?id=com.vectorunit.blue

Return to the full version of this article